Master Thesis Computing Science

Event Driven Architecture in software development projects

Author:
Maxime Klusman

Supervisor Radboud University:
Prof.dr.ir. Rinus Plasmeijer

Supervisor Sogyo:
Drs. Ralf Wolter

Radboud University, Nijmegen
April 8, 2016

Table of Contents

1. Introduction

This section gives the motivation for doing this research project and shows its relevance with respect to earlier conducted research. It then presents the research goals and chosen methods to achieve these goals. Subsequently, the scope of the project is outlined by describing some decisions that were made about what is and what is not included within this project. Finally, the organisation of the rest of this thesis is explained.

1.1. Problem statement

In computer systems, the standard communication method is request-response, where one part of the system instructs another part to perform some action or retrieve certain data by calling one of its methods [1]. The traditional way to design an information systems is to use this communication style exclusively throughout the program (for example using an Object-Oriented design) and combine it with a relational database to store the data. Alternatives to this approach are lesser known and may therefore

be overlooked, even if they would fit better. One of the possible alternatives is Event Driven Architecture (EDA) [2]. EDA is an architectural software pattern concerning the communication methods in a system. It differs from a request-response approach, like a completely Object-Oriented approach, in that it separates the system into units that communicate through events instead of method calls.

This research project was performed at software company Sogyo [3]. Sogyo has been doing software project since 1999. Since 2002, events have played a role in their development, but complete EDA was only experimented with once for an internal application. From this experiment, clear benefits of EDA were observed, which lead to the hypothesis that using EDA in the small to medium sized information systems could have substantial benefits, especially in the area of post-developmental changes and extensions to the system. However, more research was needed for Sogyo to gain enough confidence and experience to use this approach in external projects. EDA has already been shown to work well for inherently event-driven applications. For example, applications where constant input from sensors is used to make decisions and invoke actions, like in [4] and the experiment at Sogyo.

This project investigates whether the noticed benefits of EDA carry over to standard information systems, where EDA would not immediately be an obvious choice. Based on the results, members of future development projects can better assess which architecture suits their requirements the best.

1.2. Relevance

Event-driven applications have already been around for a longer time, but have recently gained more popularity [5], especially in combination with Service Oriented Architecture (SOA) for enterprise-wide applications [5, 6]. In this project, EDA is studied in a different setting. Instead of enterprise-wide applications, we look at the development of custom software for small to medium size information systems. Also, EDA is studied on its own instead of in combination with SOA. Empirical research on EDA in this setting has not been found in current literature.

1.3. Research goal

The goal of this research project is to determine the advantages and disadvantages of using EDA for a development project as opposed to a more traditional approach, and to produce some general recommendations about the use of EDA based on these findings.

Specifically, the development projects targeted in this research are those for small to medium sized information systems.

By the traditional approach, we mean an architecture where communication is done synchronously and through commands/requests and replies. Since the currently most used method is an Object-Oriented design, this is taken as the more specific traditional alternative to compare against EDA. The differences between these approaches have multiple aspects, which is distinguished in the conclusions.

1.4. Research method

As the main research method, we perform a case-study. This case-study involves the comparison between systems that are equivalent in terms of functional requirements, but use different software architectures. One has the standard, request-response (Object-Oriented), approach and the other has an EDA approach. For the request-response system, a pre-existing system made by Sogyo is used. The EDA system is designed and partially implemented specifically for this research project.

The two systems are compared on a selected set of quality attributes. The selection process involves the collection of a broad set of possible criteria and then narrowing them down by relevance until a manageable set is obtained. The selection process is described in detail in section 4.

For the comparison of the two systems on these quality attributes, quantitative measurements are combined with qualitative observations. To get conclusions that fulfil the research goals, the results are generalized where possible.

1.5. Scope

To keep an appropriate scope for this research project, choices were made about what to include and and what to leave out.

The first scoping choice is to investigate a single case study in this project. More cases would give a better foundation for generalized conclusions, but including more than one was not feasible due to time constraints for this project.

Also, for the comparison done in the case study, a selection of the most relevant quality attributes was made (see section 4), instead of considering all of them.

Another scope-limiting choice was one regarding the extend of the implementation (see section 7.3). Making an implementation that covers more of the architecture would

mean more data for a comparison and less extrapolation. On the other hand, a larger implementation would take up more time, which then cannot be spent on other aspects of the research project, so a trade-off needed to be made. Therefore, it was decided to build the implementation incrementally; the size was increased bit by bit until the point where the added value was expected to be low compared to the extra time and effort required. This way, results from the partial implementation could be generalized with high confidence, without needing to spend an excessive amount of time on the implementation.

When selecting the change-scenario's to use in the comparison (see section 8.1), some scoping decisions also had to be made. More change-scenario's would provide more comparison data. Initially, two scenario's were selected. They proved to be time-consuming, so due to time constraints, no more were added.

Finally, the theoretical research consisted of a literature study about quality attributes, background research about software architecture and most importantly, an extensive study on the topic of EDA.

1.6. Thesis outline

Since the subject of this project is software architecture, an introduction on this subject is given in section 2. Section 3 introduces EDA. Subsequently, in section 4, the selection of a suitable set of quality criteria is described.

The case study is described in sections 5-9. First, section 5 gives the general overview of how the case study is conducted and which system is used for it. Next, sections 6 and 7 give a description of the original architecture and the newly designed event-driven architecture respectively. Section 8 presents the results of the measurements made on both systems and section 9 gives an interpretation of these results and compares the two systems.

In section 10, the conclusions of the case study are generalized to give overall conclusions and make recommendations about the use of EDA in development projects. Finally, appendix A gives definitions of quality criteria used in section 4 and appendix B presents the complete measurement data.

2. Introduction to software architecture

In this section, the concept of software architecture is introduced. Additional focus is placed on how architecture can be described, since this is needed in this research

project.

2.1. History and definition

In the late 1980s, the term software architecture was introduced to denote large-scale structures of software systems. This was the result of software intensive systems becoming increasingly larger and therefore their internal structure becoming increasingly important [7]. Software architecture extended other concepts like information hiding and modularization [8].

Initially, the area of software architecture largely existed of qualitative descriptions of structural principles that had proven useful in practice. Since then, the field has grown to envelop a broad set of notations, tools, and analysis techniques [8]. Nowadays, there is also a variety of different structures that are referred to as software architecture by different people. It is a generally acknowledged fact that there is no standard definition of Software Architecture [9]. The often cited Bass et al. [10] define it as follows:

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [...] An architecture is foremost an abstraction of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements.

According to this definition, every software system has an architecture, because a system always consists of elements and interactions. This even holds in a trivial way for a system consisting of a single monolithic unit. The existence of an architecture is independent of whether it is described or documented in any way, or given any consideration in the development process. A description of the architecture and explicit consideration of its design can be of great value [10], for example when maintaining the system after its initial development. This importance of architecture increases for more complex systems.

2.2. Architectural representation

An architectural representation provides a way to describe and analyse the high-level properties of a complex system [7]. This gives developers a way to reason and communicate about these high-level properties. It is also a basis for communication with stakeholders and can serve to get a better grasp on the requirements of the system. Furthermore, the architecture often forms the foundation for the realisation of desired

quality attributes [10, 11]. When it is considered explicitly, this provides opportunities to evaluate these quality attributes to some extent early in the development process and make changes if needed. The architecture consists of properties of the system that are hardest to change later on, so explicit consideration of architecture early in the development process is very important.

2.2.1. Architectural 'views'

What was originally often contained in the concept of an architectural description, were different types of information that became mixed in the architectural representation. For example, the division of the system into smaller parts can be done in terms of the development modules (e.g. division into projects or, finer grained, division into classes or functions) or in terms of runtime units (e.g. division into applications or, finer grained, division into runtime processes) [10]. When these different types of information are mixed together into a single architectural description, it easily gets confusing. To solve this, the concept of *views* was introduced; each view represents one type of information in a separate description.

A well-known example of this separation is the 4+1 view model from [12]. The 4+1 model describes software architecture using five concurrent views. In it, the descriptions of the architectural decisions are organized around four views: logical, process, physical and development, which are illustrated with a few selected scenarios (the fifth view).

In an architectural view model, the views are never completely independent. They have certain relationships and connections, which should also be included in the overall architectural description. The architectural descriptions in sections 6 and 7 make use of a suitable set of views. A complete 4+1 view description would be excessive here, so only the process and development views from the 4+1 view model are used. The architectures are described in terms of the individual views, as well as the connections between them.

3. Introduction to Event Driven Architecture

In this section, the architectural pattern *Event Driven Architecture* (EDA) is described. The description includes theoretical strengths and weaknesses found in literature. These are used in section 4.2.2 to assess quality criteria on theoretical grounds, and in section 6.3 to make concrete hypotheses about the results of the use of EDA in the case-study.

EDA is a software pattern that can be used when designing the architecture of a software system, it deals with communication within the system. In EDA, the system is divided into a set of so-called modules. This division is primarily on an executional level, dividing the system into separate runtime units. Between these modules, the communication is achieved by event notifications ('events'). These events often take the form of messages. An event is a record of something notable that happened within the scope of the system [2, 6]. It is generated by an event source and broadcast into the system [1]. Each module can define which events should be acted upon. In the interested modules, the event triggers some action, while other modules simply ignore it. The nature of events is descriptive; they convey what happened, but do not in any way specify what actions should be taken as a result, or by which or how many modules [1].

Modules do not have any information about each other. The principles of EDA dictate that a sending module should have no knowledge of the way its events are processed. The knowledge that *is* shared between modules, is about the events themselves: the types of events that are sent in the system, what kind of information they contain and in what form this information is present. This has some consequences. First of all, it makes it possible for the modules to be extremely loosely coupled [2, 6], which is often a desirable quality. Also, it makes EDA work very well with asynchronous flows of work and information [2]; if receiving module(s) of an event are busy, there is no inherent need for the sending module to wait. The advantages of this are that modules have to spend less time waiting and that there is potential for horizontal scaling; work could be divided between different copies of the same module. On the other hand, it decreases the amount of information and assumptions that can be made within modules. Additionally, some amount of synchronization may be required in a system, which then needs to be handled explicitly in an EDA system.

The EDA set-up also means that new information is pushed around the system rather than provided on demand. Consequently, updates are made in reaction to changes, which is useful in many systems. It means there is no need to poll for any changes elsewhere in the system, which can improve timing properties. On the other hand, it also means modules that need information on demand, need to keep a copy of it themselves. This increases the overall storage load.

Finally, in an EDA system, actions are less sequential. This means it is suitable for situations where a change has multiple consequences, which can be processed more or less independently of one another. It also means the total behaviour of the system is likely to be more difficult to analyse, since the order in which things happen is less defined and can differ between equivalent runs of the system.

4. Selection of quality criteria

This section describes the process of selecting a suitable set of quality criteria to use when comparing the two case study systems. The following steps are used in this process:

(a) collecting a broad set of possible quality criteria from literature,
(b) narrowing down this list to a feasible set of relevant criteria,
(c) defining the way each remaining criterion is measured.

Section 4.1 defines the result from step (a) in the form of a long-list of quality criteria. Section 4.2 follows up with an explanation of the process of step (b). The result of this step is shown in the form of a short-list in section 4.3. Finally, section 4.4 elaborates on step (c).

4.1. Long-list of quality criteria

The starting point is a list of possible criteria that can be used in general to determine the quality of software and software architecture. To make this list, information about quality attributes that can be used to evaluate software systems and architectures is collected from the following sources: [13, 14, 10, 15, 16, 17, 18, 19, 20, 21]. The collected information is then combined and aggregated to produce a custom grouping, shown in table 1. Only the entries marked $^{(*)}$ are included in the short-list. The rest are excluded for various (numbered and color-coded) reasons, which are further explained in section 4.2. Table 18 in appendix A shows the definition of each quality attribute, as well as the direct source it was adapted from.

The quality criteria, their grouping, and their definitions are already somewhat modified towards convenient use in subsequent steps, but are still included in a way that is relatively close to the source materials. As in the source materials, a group of sub-criteria does not always form a complete description of its main criterion. Also, some criteria have overlapping definitions. For example, Maintainability contains a sub-criterion of Evolvability, which overlaps with Modifiability and Extensibility. Also, the notion of Scalability, the ability to improve a system's performance by adding more resources, is part of Maintainability but missing from the list of sub-criteria. For the short-list (see section 4.3), the remaining criteria are further modified to make them more suitable for measurement on real systems.

(1) **Functionality**	(2) **Usability**	(4) **Portability**
(1) Traceability	(2) Operability	(4) Machine Independence
(1) Suitability	(2) Appropriateness Recognizability	(4) Replaceability
(1) Appropriateness	(2) User Error Protection	(4) Installability
(1) Correctness / Conformance	(2) Attractiveness	(4) Adaptability
(1) Completeness	(2) Communicativeness	(4) Software System Independence
	(2) Learnability	
(5) **Security**	(5) **Reliability**	(*) **Maintainability**
(5) Accountability	(4) Robustness	(*) Testability
(4) Authenticity	(3) Maturity	(*) Modifiability
(4) Integrity	(3) Accuracy / Precision	(*) Extensibility / Augmentability
(5) Non-repudiation	(5) Fault Tolerance	(*) Repairability
(4) Confidentiality / Privacy	(5) Recoverability	(*) Evolvability
	(5) Availability	
(5) **Compatibility**	(5) **Reusability**	(*) **Audit trail**
(4) Co-existence	(5) Generality	
(5) Interoperability	(5) Self-containedness	
(*) **Efficiency**	(*) **Structuredness**	(*) **Understandability**
(5) Capacity	(*) Consistency / Uniformity	(3) Legibility
(*) Performance / Timeliness	(*) Conciseness	(*) Analyzability
(*) Resource Behaviour	(*) Modularity	(*) Simplicity
		(*) Self-descriptiveness

Table 1: Long-list of quality criteria

4.2. Exclusion of criteria

A portion of the quality criteria are excluded for the comparison performed in this research project. The different reasons for exclusion are numbered and color-coded in table 1.

First of all, the functional criteria marked (1) are excluded. This is explained in section 4.2.1. Additionally a number of non-functional criteria are also excluded. They are first filtered on a theoretical basis; for each criterion it is determined whether or not it is affected by a change from a request-response architecture to EDA. This is done based on the following three arguments:

(2) Criteria that involve Usability focus on the user experience. This only involves the external behaviour of the system and is therefore independent from the architectural structure [22]. Hence these criteria are irrelevant for the comparison that we want to make in this project.

(3) A few criteria consist only of properties of the implementation (choices made on the programming level) and are therefore independent of the architecture.

(4) Criteria may be dependent on whether the architecture has specified certain features, but if these features are independent of the internal structure, they are not relevant for the current comparison; they could be added to or removed from a system with either architectural structure.

The exclusion of criteria based on (2) and (3) is fairly straightforward and is not further explained. The assessment of the remaining criteria regarding (4) is explained in section 4.2.2.

Finally, some of the remaining criteria are of less interest to Sogyo with respect to this research topic. These are excluded to keep a manageable scope for this project, and are marked (5) in the table. The considerations that lead to these choices are clarified in section 4.2.3.

4.2.1. Functionality

In this project, functionality plays a slightly different role than the other, *non-functional*, quality attributes. Bass et al. [10] describe the relationship between functionality and architecture as follows:

> "Functionality may be achieved through the use of any of a number of possible structures. In fact, if functionality were the only requirement, the system could exist as a single monolithic module with no internal structure at all.

> Instead, it is decomposed into modules to make it understandable and to support a variety of other purposes. In this way, functionality is largely independent of structure."

Similarly, the functionality that can be provided by the use of the principles of either EDA or request-response, can be shown to be the same. The most fundamental difference between the two is the semantics, which is mainly reflected by how things are named. However, these choices only influence the way people think about and understand the concepts used in a system and do not limit the possible functionality in any way. Another, more concrete difference that often (but not necessarily) occurs between the two principles, is synchronous versus asynchronous communication. EDA combines well with asynchronous communication, because the sender is not responsible for how, where and when the event is received and can usually continue execution as soon as it has given off its event notification to the underlying framework. These two communication methods can also be shown to be equivalent. As explained by [23], asynchronous communication can be modelled by synchronous communication by introducing a buffer. This buffer means the message can be sent at any moment by the sender, and is stored in the buffer. The receiver can collect it whenever it is ready to, without the sender having to wait. The other way around, as mentioned by [24], synchronous communication can be modelled by hand-shaking of asynchronous processes. This means that in terms of functionality, the two principles are interchangeable.

A formal method of conversion from request-response to EDA is unlikely to provide major benefits in terms of *non-functional* quality attributes. To achieve real benefits, the architecture needs to be reconstructed with an event-driven mindset. Along the same lines, Bass et al. [10] add:

> "Software architecture constrains [the allocation of functionality] to structure when other quality attributes are important. [...] The interest of functionality is how it interacts with, and constrains, those other qualities."

This view corresponds to the importance of functionality in this project. It does not lie in how well functionality requirements can be achieved through different architectures, but rather it lies in the way non-functional quality attributes are influenced by the architectural method chosen, when constrained by the functional requirements. For the purpose of this project, this means the functional attributes from table 1 are not appropriate as quality criteria. Instead, functional requirements being satisfied is a prerequisite to start assessment of non-functional criteria for the architecture.

4.2.2. Theoretical assessment of quality criteria

The following paragraphs explain for each criteria group the reasons why the contained criteria are or are not excluded on theoretical grounds.

Portability and Compatibility

All criteria in the Portability group have to do with the system's independence from its environment. The same holds more or less for the Compatibility criterion Co-existence; this criterion restricts the way the system influences parts of its environment and resources that it shares with other systems. The environment that is meant here is the environment of the system as a whole. This environment is for example an operating system or a web browser. The mentioned criteria view the behaviour of the whole system with respect to factors that originate from outside of the system.

On the architectural level, satisfying the above mentioned criteria requires mechanisms, that act as a layer between the core of the system and some part of its environment, to be specified by the architecture. Examples of such a mechanism for the criterion Adaptability are genericity and parametrization [22]. The mechanisms depend on technical details of the environments and the system itself. They are not dependent on the principle behind the communication method (events versus request-response). This is because the communication method between the layer around a part of the environment and the part of the environment itself does not qualify as communication between modules of the system. Therefore it is not restricted and can be the same in both architectural methods (EDA and request-response). The communication method used on the boundary between the layer around the environmental component and the rest of the system *is* restricted, but here the only thing that is important is the content of the information that is exchanged and not the semantic form. Since both communication method are able to convey the same information (see section 4.2.1), they are equivalent in this regard. Therefore the criteria in this group do not provide a good basis for comparison of these architectural communication principles.

The same argument does not hold for the Compatibility criterion Interoperability. Here, domain information needs to be exchanged between systems. The technical details of the information exchange are in principle independent of EDA. However, the exchange of information between systems may be linked to the internal exchange of information of the system. If a pair of interoperating systems do not share a communication principle (either event- or request-response-driven), a semantic translation may be needed before the systems can communicate effectively. On the other hand, if both systems adopt EDA, the communication could be achieved with relatively little effort by listening to one another's events. This criterion therefore *is* theoretically relevant in the comparison

between communication methods in this project.

Security

Some features of Security (Non-repudiation, Accountability) rely on the ease of keeping an audit trail, discussed above.

Authenticity and Integrity, on the architectural level, require a mechanism for authentication to be present which verifies that entities are who they say they are. This mechanism can be completely encapsulated within a single module and is therefore not dependent on communication method between modules. This makes the quality criterion Authenticity irrelevant for this project. Integrity additionally requires the system to keep track of the verified identity of an entity and provide or deny access to certain features based on it. This relies on implementation details, but can also be dependent on communication between modules. For example a module may be required to know which user is logged in to provide or deny access to some data. However, this information can be conveyed with either communication strategy, so this criterion is also not relevant for the comparison. Confidentiality / Privacy can be seen as a sub-criterion of Integrity, so the same reasoning applies.

Reliability

The Reliability criterion Robustness is concerned with the behaviour of the system under conditions that are not explicitly specified by the specification. This means, for example, how the system handles inappropriate user input. Even if these conditions are not explicitly specified by the specification, the way they are handled in the architecture is the same as for specified conditions. Therefore, where architecture is concerned, Robustness can be treated as a sort of extension of functionality and is independent of communication strategy. As a result, this criterion is excluded from the list.

The Reliability criteria that have not been excluded up to this point, *are* dependent on communication method. This is because in a system with EDA, redundancy can be achieved by having multiple run-time modules that provide the same functionality (implemented either in the same or a different way). None of the other modules need to be changed because of this. If one of these redundant copies fails, the other(s) can provide the functionality to keep the system running correctly. Also, if all events in the system are recorded, the system state can be recovered after a failure by reconstructing it from these events [1].

Efficiency

The Efficiency criteria are all dependent on the use of EDA. A system with EDA is well suited for asynchronous communication and therefore also for parallelization [2]. Both can greatly enhance performance.

Additionally, with EDA, state keeping is usually handled in a significantly different way.[1] Instead of requesting data from one central source each time it is needed, all modules that need access to certain data keep a local copy. Whenever the source experiences a change, it generates a corresponding event. All modules that keep a copy of the affected data, respond to the event by updating their copy. Whenever these modules need access to a piece a data, they can access it locally. This form of redundancy means that the resource usage for an EDA system is higher. The performance and capacity are higher, because modules never have to wait for response from the source. Therefore the data source never becomes a bottleneck even when many other modules need its data at the same time.[1]

Audit trail

For a system with request-response, setting up an audit trail during run-time of a system requires extra logging effort. In a system with EDA, it is easy to track and analyse all interaction between modules by capturing all events that are created in the system [1]. This information can be used on its own or in combination with other logging information. This means this criterion is dependent on the communication method and therefore theoretically relevant in the comparison.

Structuredness

EDA demands a certain degree of Structuredness. The division of the system into modules that only communicate through events, makes the modules very loosely coupled [2, 1], which provides a high Modularity. It also provides top-level architectural Consistency. Conciseness could potentially also be heavily influenced by the choice in communication strategy, because certain functionality may be expressed more easily and with fewer code in one strategy compared to the other. These criteria are therefore all relevant in the comparison.

Understandability

All main architectural choices naturally influence the Understandability of the whole system. For EDA specifically, the control flow may be harder to distinguish, since in the source of an event, it is not known what responses it triggers [2, 7]. On the other hand, in

an EDA system it is easier to track and therefore analyse all interactions that have taken place between modules (see Audit trail, page 17). Also, events tend to be descriptive, so they inherently convey information about their purpose. Understandability criteria (aside from Legibility, which is defined on the implementation) are therefore relevant in the comparison.

Maintainability

Maintainability concerns the ease for a system to be changed with respect to different aspects. The reparation of faults (Repairability) and changes in functional features (contained in Modifiability, Extensibility, Evolvability) are mainly dependent on the criteria described in Structuredness, page 17 and Understandability, page 17.

The ease of verification of changes in the system (Testability) is also dependent on Structuredness because highly independent modules can be tested separately. Additionally, it is dependent on the ease of keeping an audit trail (Audit trail, page 17). This is because test cases can easily be constructed from recorded sequences of events. It is further dependent on the choice in communication method, because EDA allows for less static validation. As a result less errors can be found at build-time, so they have to be found through run-time testing.[1]

An increase in desired capacity or throughput of the system (contained in Extensibility, Evolvability) is dependent on parallelization and possibility for distribution, which are both influenced by the chosen communication strategy [2]. Together this means all of the Maintainability criteria are relevant in the comparison.

Reusability

Reusability is influenced by aspects of Structuredness (page 17), because loose coupling improves reuse [1, 14]. Part of this is because any module can be introduced in an event-driven system simply by registering it for the system's events [7]. Reusability is also dependent on the way the functionality is distributed over several modules, which is influenced by the communication strategy. Therefore, these criteria are theoretically relevant in the comparison.

4.2.3. Relevance exclusion

Of the remaining criteria, some are more relevant to this research project than others. For Sogyo, the main importance for this research topic lies in the possible improvement in the criteria in the Maintainability group. Nowadays there are often rapid changes in user

requirements as well as technology and environment. This means that a software system is continuously being changed throughout its lifetime. It is therefore very important that a system can be relatively easily modified to accommodate these changes, while maintaining architectural integrity. If the integrity is not maintained, the cost of further modification increases. This makes maintainability aspects an important concern for virtually every software system.

Additionally, there is an interest in Efficiency aspects. EDA has a very different approach to information distribution and storage compared to traditional request-response systems that often rely on a central relational database. It is difficult to predict how much this difference in approach influences the time and resource behaviour. Specifically for the case-study project, there are issues with reaction times, which may be solved by an EDA approach.

Since these mentioned criteria are together heavily dependent on the criteria in groups Structuredness, Understandability and Audit Trail, the criteria in these groups are also kept. The other criteria are not explicitly evaluated in order to keep a manageable scope for this research project.

4.3. Short-list of quality criteria

When all excluded criteria are removed from the long-list, a short-list is obtained. These remaining criteria are used to compare both architectural approaches in the case study. Some criteria are redivided in a more complete and less overlapping set of sub-criteria. Since several criteria are dependent on other criteria, a graph-like structure is appropriate for the representation of this list. This representation is shown in figure 1. In this figure, $A \longrightarrow B$ means that A influences B, i.e. B depends on A. The dashed arrows mean a conditional form of influence; time and resource behaviour depends on scalability only when the system actually needs to be scaled up. The definitions of all the (possibly modified) remaining sub-criteria are shown in table 2. In the reformed structure, the set of criteria is better suited for the next step of the research process: the definition of measurement methods for the different criteria.

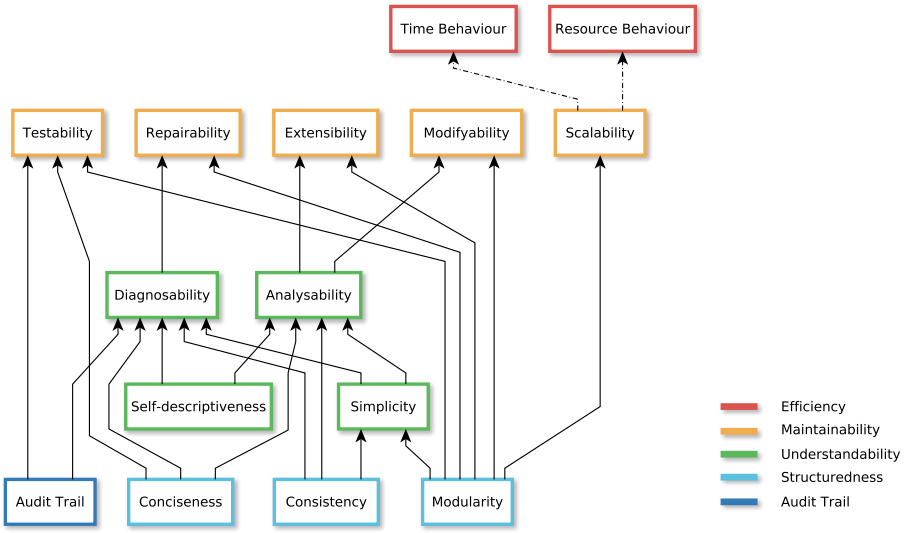

Figure 1: The short-list of criteria, in graph from to show dependencies

Time Behaviour	degree to which the response and processing times and throughput rates of a system, when performing its functions, meet requirements
Resource Behaviour	degree to which the amounts and types of resources used by a system, when performing its functions, meet requirements
Repairability	ease with which errors can be located and repaired in an operational program while maintaining architectural integrity
Modifiability	ease with which existing features can be changed to suit changes in requirements in an operational program while maintaining architectural integrity
Extensibility	ease with which new features can be added to meet added requirements in an operational program while maintaining architectural integrity
Scalability	ease with which the system can be changed to accommodate and increase in scale: more data / users / nodes, while maintaining architectural integrity
Testability	ease with which test criteria can be established for a system or component and tests can be performed to determine whether those criteria have been met

Diagnosability	ease with which deficiencies can be diagnosed and causes of failure can be located
Analyzability	ease with which the parts to be modified for an intended change can be identified
Self-descriptiveness	extend to which objectives, assumptions, constraints and in-/outputs can be inferred from the code alone (without comments)
Simplicity	ease with which the overall structure (elements and interactions) and the control-flow of the system can be understood
Audit Trail	degree to which actions performed during the system runtime can be reproduced
Modularity	degree to which changes in one module have the least impact on other modules
Consistency	degree to which uniform design and implementation techniques and notation are used
Conciseness	degree to which the implementation of functionality is provided with a minimum amount of code

Table 2: Custom definitions of short-list quality criteria

4.4. Methods of measurement

In order to compare different architectural approaches based on the selected criteria, these criteria need to be measured. To this day, there is no evaluation method that is completely objective, extremely precise and also requires very little effort to use [11]. Therefore, a set of appropriate techniques that give a reasonable balance between accuracy and effort, are chosen. This section briefly describes the different chosen techniques.

Some methods include metrics that are measured on the implementation, either on the code base or on the running system. Since the alternative solution using EDA is only partly implemented, a comparable set-up for both systems needs to be created. This is done by selecting part of the functionality and creating set-ups for both versions of the system that include only the parts that are necessary to provide the selected functionality.

4.4.1. Efficiency

The most reliable way to measure both time and resource behaviour, is by measuring these quantitatively during runtime of both versions of the system.

The time behaviour is assessed by specifically measuring the completion time of a set of user actions. Since all user actions for the selected system are server requests, the completion times are measured in terms of loading time of the request. These loading times are measured with a browsed-based tool called HttpWatch (Basic Edition 10.0.20) [25].

The resource behaviour is assessed by manually calculating the total amount of used RAM and disk space used by both methods at different characteristic times during runtime of the system.

4.4.2. Structuredness and Understandability

The below mentioned metrics on the source code is calculated using Visual Studio Code Metrics Powertool. The results are aggregated and viewed in Code Metrics Viewer.

- Conciseness is measured by the metric *Lines of Code*.
- Modularity is measured by the metric *Class Coupling*.
- The *Maintainability Index* gives an indication of Diagnosability and Analyzability.
- *Cyclomatic Complexity* and *Depth of Inheritance* is used to give an indication of Simplicity.

4.4.3. Audit trail

For the ease of creating a recollection of notable things that occur at runtime, no real quantitative measures were found. Instead, a qualitative comparison is made, which describes the points of difference between the two approaches.

4.4.4. Maintainability

For the assessment of Modifiability and Extensibility, a scenario-based approach is used. This involves the construction of several change-scenarios and evaluation of the amount of effort it would take to apply these changes to the system. The approach is based on the methods described in [26], [27] and [28]. The actual scenarios that are used are based on actual changes that have happened in the original system, so that they are as

realistic as possible. To determine these scenarios and the amount of work they took in the original system, the source control repository is used.

Scalability is determined by analysing how the system's behaviour changes depending on size characteristics. The time and resource behaviour measurements are used for part of this analysis.

In accordance with figure 1, the results for some of the other quality criteria are also used and combined to make statements about Maintainability aspects.

5. Case study: Set-up

In this section, the case study is introduced. Section 5.1 describes the way the case study is conducted. In section 5.2, a brief description of the background of the used case is given, along with the general functional requirements.

5.1. Way of working

This case study provides a comparison of two systems that fulfil the same functional requirements, but have different underlying architectures; one has a traditional architecture and the other is event-driven. This comparison is then used to make more general statements about the advantages and disadvantages of EDA.

The starting point of the case study is an existing solution that was previously designed and built by Sogyo. We call this existing solution the *original system*. The original system takes a more traditional approach and is not event-driven. Section 6 describes the architecture of this original system.

With the same functional requirements as the original system, an alternative solution is designed using the EDA pattern. This alternative solution is called the *EDA-system*. Part of the designed EDA-system is also built as an application, using an existing event-framework that was previously made by Sogyo for the purpose of experimenting with EDA. By actually implementing part of the EDA application, the practical lower-level implications of the top-level design can be seen. Additionally, the implementation can be used for (quantitative) measurement. Section 7 describes the architectural design of the EDA-system.

Finally, the original system and the EDA-system are compared using the quality criteria selected in section 4. The results of this comparison are presented in section 8.

As introduced in section 2.2.1, the structure of a system's architecture can be described in different *views*. To make comparison between the two systems easier, their architectures are described using the same set of views. The used views are the following:

- The *process view* shows the division into runtime execution units that run in parallel, like applications, processes and/or threads.
- The *development view* shows the division of the system into development units like libraries, projects, namespaces and/or classes.

5.2. Case description

The chosen case is a system requested by a company (the client) that sells products to stores, which in turn sell them to consumers. These stores are called the company's *customers*. Contact with a customer is handled by a *sales person*. The customers are divided into areas and each area is represented by one sales person. This person maintains the relationship with the customer and establishes which products they want to buy. This is captured in so-called *budget proposals*. Budget proposals are made once a year; each is a signed agreement with a customer, which describes what products the customer will buy in the upcoming year. Additionally, a sales person tries to enlist new customers.

One of the main purposes of the system is to assist each sales person in their sales process. A sales person wants to be able to view comparisons between the number of sales estimated in budget proposals and the actual turnovers for the same customer and time periods. This helps them make more accurate budget proposals for upcoming years. The system is also meant to provide the *director* and *area managers* with data to support decisions and changes in policies.

5.2.1. Functional requirements

The system takes the form of a SalesPortal, which provides a set of features to users according to their role. There are four available roles: sales person, area manager, director and administrator. The global actions they can each perform using the SalesPortal, are:

- A sales person can (1) enter new customers into the system, (2) view different kinds of data about their customers (contact details, characteristics, budget proposals, actual turnovers, etc.), and (3) make budget proposals.
- An area manager can view different summaries about the status of business in his/her areas.
- The director is able to see summaries about the entire set of areas.

- The actions for the administrator are (a) management of SalesPortal users, (b) correcting the imported data, and (c) checking whether the system functions correctly.

The system should additionally be able to process input data from an external system (in this case SAP), which the client also uses to support their business processes.

6. Case study: Original architecture

This section gives a description of the original system's architecture. As described in section 5.1, the architectural description is divided into a *process view* (section 6.1) and a *development view* (section 6.2). Additionally, section 6.3 presents the weak points of the original system and hypothesize about how EDA could improve upon them.

6.1. Process view

The *process view* shows the division into runtime execution units. A schematic representation of this view for the original system is shown in figure 2. In this figure, the green entries are separate runtime execution processes and the orange entries are forms of data storage and exchange.

In this view, the system is divided into an database server (Microsoft SQL Server) and three separate applications that connect to it. The way this connection is handled, is described in the *development view*. Two of these applications, the batchimport and the batchpdfcreator, are processes that are run overnight at predefined times. The third is the web application that provides the interaction with the users of the system. The web application is hosted on an ISS Express server and an arbitrary number of users can connect to it to use the system.

The batchimport process handles the import of data from SAP. This data is presented in the form of XML files, which are read in by the batchimport application and stored in the database.

The batchpdfcreator uses data from the database to generate a set of PDF and HTML tables. These are stored on disk and used by the web application to include in certain web pages that are sent to the user. This means some tables in the web application are pre-generated by the batchpdfcreator. However, there are also tables in the web application that are not generated by the batchpdfcreator, but instead are generated on demand (pull-based).

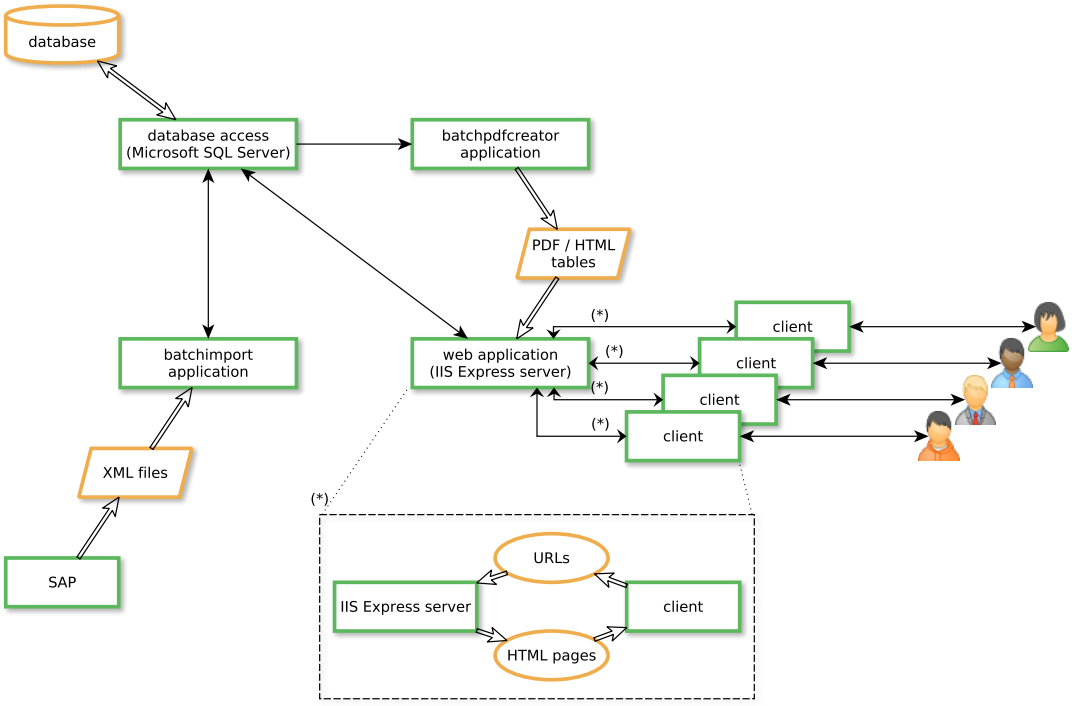

Figure 2: Original system – schematic representation of the *process view*

The web application is constructed using ASP.NET MVC, further described in the *development view*. The client-server interaction is shown in more detail in the (*) labelled dashed box in the figure. It is handled in the following way:
A user specifies the URL of the page they want to view, either by clicking a hyperlink or typing it in manually. The URL specifies which action on which controller on the server should be executed. This action results in the creation of an HTML page that is sent back to the client and shown to the user.

6.2. Development view

The *development view* shows the division of the system into development units. A schematic representation of this view for the same system is shown in figure 3. In this figure, the dashed boxes are separate projects / libraries. Dark blue represents the developed software, while light blue represents imported third-party libraries. The red entries denote relevant C# classes to show connections between the projects.

First of all, the batchimport and batchpdfcreator projects in this view directly correspond to the applications of the same name in the *process view*.

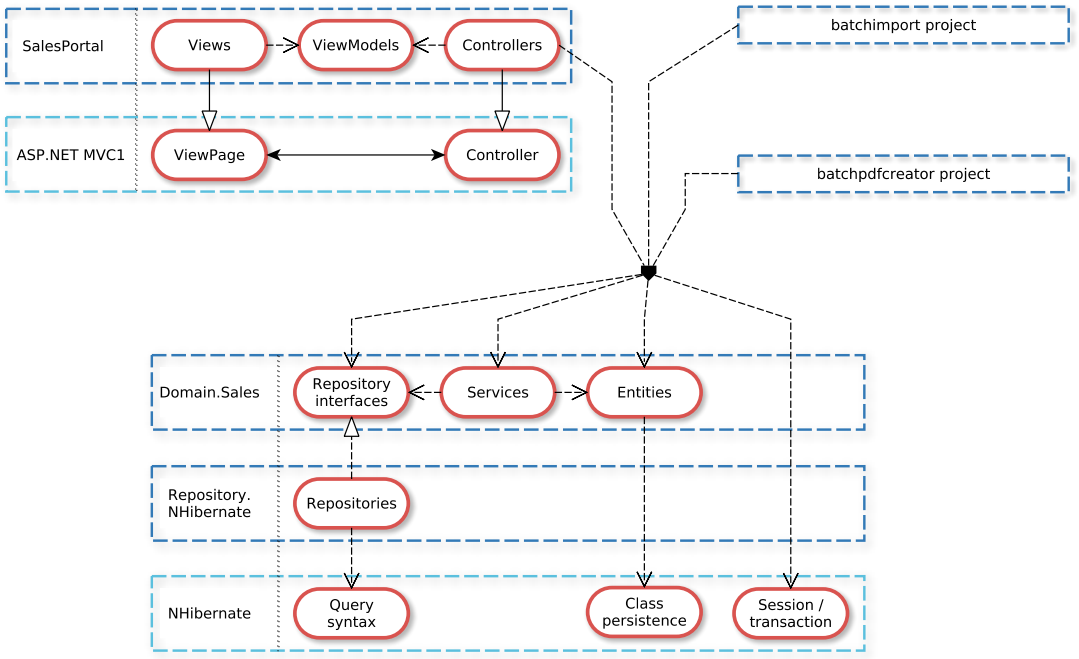

Figure 3: Original system – schematic representation of the *development view*

Additionally, this view shows how the ASP.NET MVC library is incorporated into the SalesPortal project to construct the web application. The MVC library provides base classes for the Controllers and the Views. Each controller implementing this class, can define a set of actions that are initiated by the client by navigating to the corresponding URL. These actions each connect to a view, which defines the HTML page that is constructed and sent back to the client. The connection between controller and view is provided by the MVC library. The ViewModels can be used as an easy way to transfer data from the controller to one of its views. They are called ViewModels instead of just Models (as they are called in MVC) because they do not correspond to the actual domain entities. Instead, they correspond with how the data is organized as it is shown to the user.

Finally, the three mentioned projects make use of a set of layers that handle the connection to the SQL database. The top layer consists of the sales domain. It contains classes for the entities that exist in the system domain, defines a repository interface for each of these entities and also defines a service for some of the more important entities.

The entity classes are directly linked to the database by using the class-persistence functionality of the NHibernate library [29]. This functionality provides *get* and *set* operations on specified attributes to be automatically translated to database queries.

The repository interfaces define a set of queries to be performed on the collection of records of their corresponding entity type. These repository interfaces are realized by a set of actual repositories in the Repository.NHibernate layer. They use the functionality from the NHibernate library to perform these queries on the database. The services use multiple repositories to provide more complicated queries on the data. Finally, the NHibernate library also contains functionality to define sessions and transaction with respect to the database. This functionality is directly accessed by the other projects for initialization purposes.

6.3. Problems and Hypotheses

The original system functions adequately, but there are a few issues that could be improved upon. This section highlights these issues and speculate as to how an event-driven solution would improve them.

6.3.1. Timeliness and Scalability

One of the main problems in the original project, is that some user actions take a long time to complete, in the order of seconds. For users, the tolerable waiting time for information retrieval is approximately 2 seconds [30], and this is exceeded in some places. The user actions that take a long time are actions where a set of data needs to be displayed (usually in a report/table) and this data needs to be collected at the time of the request. Since it involves large amounts of data that are distributed over multiple database tables, this takes a while to complete. While this happens, the user is waiting for their report to load.

Additionally, the batchpdfcreator process takes over two hours time to complete every night (in previous versions even longer). The batchpdfcreator creates a set of pdf and html reports overnight. These reports are built from scratch every time and the old set of files is overwritten. Again, a large amount of data needs to be collected and combined. Here, the user is not waiting while these actions are completed, but another timing issue arises. The batchpdfcreator process can only be started once the batchimport process has been finished, and the import in turn can only start once the XML files from SAP are available. The web application is offline during these processes, so the longer the combined task takes, the higher the risk that it has not finished yet at the time the users want to start using the web application again in the morning. Both these issues have recently been improved upon, but the performance is still not optimal. They both get increasingly worse as the amount of data grows over time.

An event-driven approach could solve both issues by saving a copy of the data necessary for each report in separate modules. The main data storage managed by a main module. This module generates events once some part of the data changes, and all affected reports are updated by their respective modules in response. This first of all means that the data does not need to be collected over and over. Consequently, the time it takes to generate the reports is not affected by the total amount of data in the system. Secondly, the processing of data, and updating of reports, is done as soon as new data is available rather than at the time a report is requested by the user.

6.3.2. Maintainability

Another thing that could be improved is the ease with which certain changes in the requirements can be implemented. The client makes these changes relatively frequently and they usually involve the change or addition of some columns in a set of reports shown by the system. Sometimes these changes can be made by simply making some changes in the ViewModels and Views of the web application. However, sometimes the changes are made by changing the domain and the structure of the database, which involves at lot more work. Also, some reports are created overnight and loaded from a file and others are created by the web application itself. If a report needs to be changed, it is not immediately clear where the change should be made.

This could be improved by using the same EDA set-up mentioned for the previous issues. The main data storage should be kept as general as possible and save as much information as is available. The way the information is displayed, can then be changed by changing only the module corresponding to the involved reports, letting them respond to different events if necessary. This means the domain would only need to be changed if the types of data provided to the system change, and not when the data is used in a different way. This would also mean that all reports are made in the same way, so changes would require similar actions for all reports. Since modules are separate parts of the system that only communicate via events, the addition or removal of 'report'-modules would not affect the data-storage module, input modules or other 'report'-modules in any way.

7. Case study: New architecture

This section describes the design of the alternative architecture. The architectural description is again divided into a *process view* (section 7.1) and a *development view* (section 7.2). Section 7.3 describes the extent of the built implementation. The different modules in the architecture are elaborated upon in section 7.4. Finally, some noteworthy

things encountered during the design process are explained in section 7.5.

The alternative architecture uses the EDA pattern as a basis. As motivated in section 6.3, the division into modules is as follows.

- one central data storage and management module (the *EventStore* module)
- one input module for the data from SAP (the *SAPImport* module)
- a *ViewModel* module for each type of report
- one 'front end' module (the *Gateway* module)

7.1. Process view

A schematic representation of this view for the EDA-system is shown in figure 4. Just as in figure 2, the green entries are separate runtime execution processes and the orange entries are forms of data storage and exchange. The four different (types of) modules in the system are highlighted by four dotted boxes with different colors. These boxes in the same colors are also used in the schematic representation of the *development view* (figure 6) to show the connection between the two views.

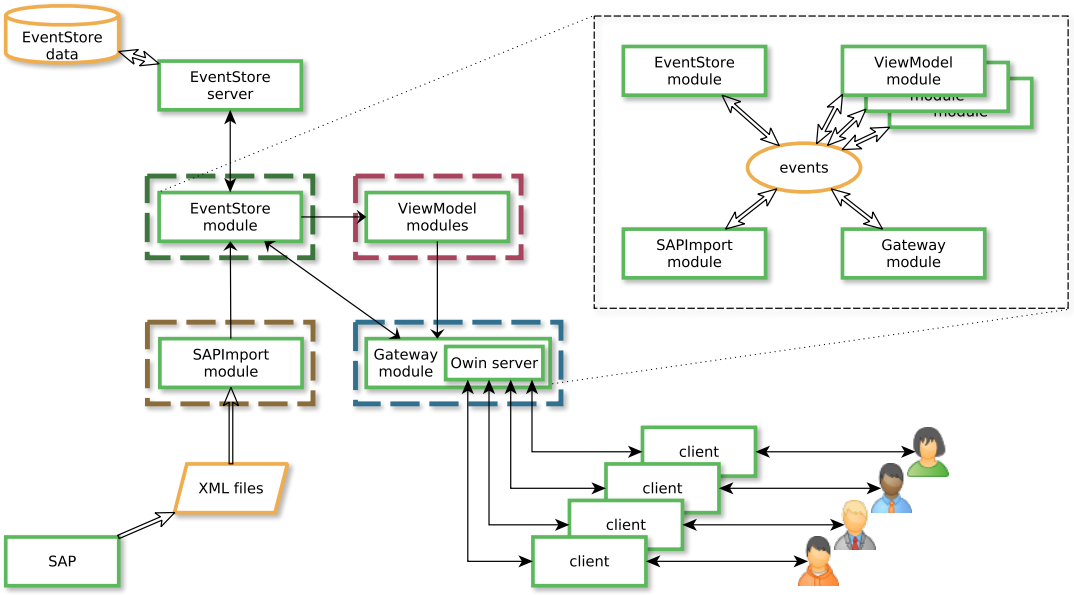

Figure 4: EDA-system – schematic representation of the *process view*

The core of the system, the modules that communicate via events, are shown in two different representations. The representation in the dashed box shows how all modules send their events into the system and all other modules can receive them. The representation outside the box shows which modules actually exchange information. For example, there

is an arrow from the EventStore module to the ViewModel modules. This means there are events of certain types (in this case the types "*Entity*Changed") that are sent by the EventStore module and received and reacted to by one or more of the ViewModel modules.

7.2. Development view

A schematic representation of this view for the EDA-system is shown in figures 5 and 6. Just as in figure 3, the dashed boxes are separate projects / libraries. Dark blue represents the developed software, while light blue represents imported third-party libraries. The red and orange entries denote respectively relevant C# classes and JavaScript scripts, to show connections between the projects.

The used Messageframework was previously developed by Sogyo. It contains base classes with which message sending and receiving can be implemented.

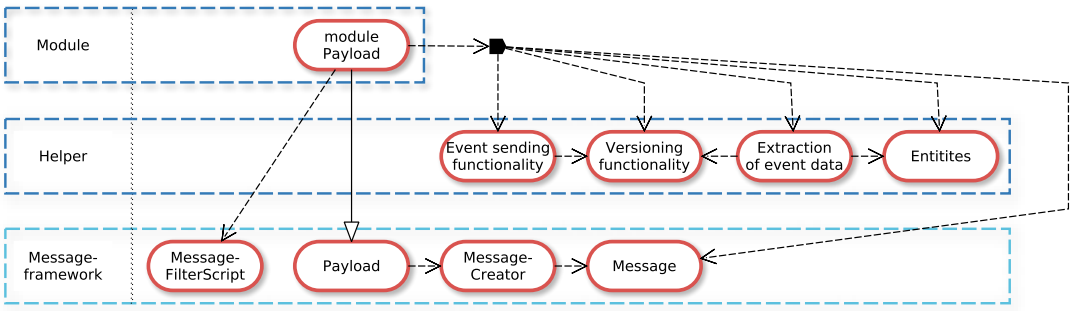

Figure 5: EDA-system – schematic representation of the connection between the modules and the Messageframework in the *development view*

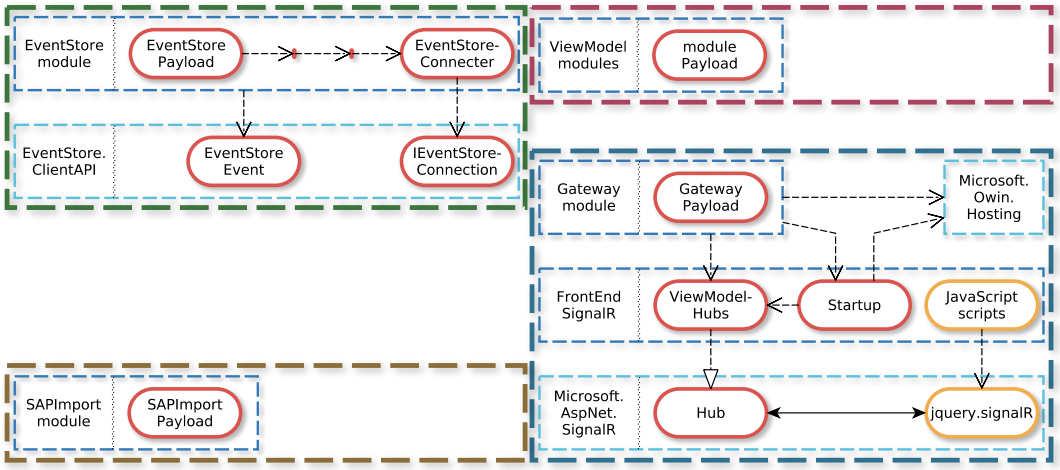

Figure 6: EDA-system – schematic representation of the modules in the *development view*

In figure 5, the connection between the modules and the Messageframework is shown. Each module contains a Payload class that inherits from the abstract Payload class in the Messageframework. The framework also provides a Message class that represents the events that are sent between modules. Additionally, the MessageCreator class is used to create and send events and the MessageFilterScript can be used in module Payloads to receive events.

The knowledge that is shared between all modules is contained in a separate Helper library. It contains for instance representations of the Entities the way they are contained in events. It also contains the different event-types that exist in the system and methods to add and retrieve specific information from events.

Figure 6 shows the four modules and their dependence on other libraries. As mentioned in the *process view* paragraph, a set of coloured dotted boxes is used to highlight the different modules.

The EventStore module uses the EventStore.CientAPI library to access an EventStore and store and retrieve data. For the Gateway module, a separate FrontEnd project was developed that uses the SignalR library [31] to establish a connection between a Microsoft Owin [32] server and JavaScript code on the client. This FrontEnd project, as well as the Microsoft.Owin.Hosting library, are used by the Gateway module to run an Owin web server.

7.3. Extent of implementation

Some details of the architecture described in the previous section, are derived from the implementation process. For example, the choices of event-framework, server etc., were all made for practical reasons. These choices are reflected in the libraries used and shown in the *process* view. This also means that the parts that were not implemented are less detailed in the architectural description given in this thesis.

The implementation covers first of all the core of the system, which consists of all functionality for sending and receiving events and the storage, front-end and SAP import modules. Additionally, it contains two ViewModel modules: one for an area overview and one for a customer overview per area. All other ViewModels modules are omitted, because their implementation was expected to be analogous to the already implemented ViewModel modules. Another thing that was not implemented, is all functionality regarding accounts, logins and user roles. This functionality required substantial additional research and is relatively independent of the event-driven characteristics of the system. In the just mentioned implemented functionality, some of the domain model entities do not play any role and are therefore also not included in the system (see section 7.4.1).

7.4. Module descriptions

The different modules are explained in more detail in the next few sections.

7.4.1. EventStore module

The central data storage and management module receives any external changes to the EDA-system as events and stores the information in a persistent way. As storage mechanism, an *event store* is chosen. This has the advantage that instead of just the current state, the entire evolution of the state is saved and can be used for various purposes. In addition, the information from incoming events can be stored in a semantically similar way, so no complicated transformations are needed.

The used event store implementation is called EventStore [33]. It contains a client API for C# so that it can be interacted with from within the program.

The incoming events for this module denote the creation, change or deletion of a set of entities that are present within the domain of the EDA-system. These entities, with the respective actions that can be performed on them, are shown in table 3. Some additional entities are processed by the original system, but left out in the implementation of the EDA-system because it was scaled down to a smaller version in which these entities did

not play a part. These left out entities can be seen by comparing figures 9 and 11 in section 7.5.3. The concept of User, which is similar to an entity, was also left out in the EDA-system implementation.

For BudgetProposals and Turnovers, the create actions are not defined. These entities have a periodic aspect and have to do with monetary values. To the outside, an entry for each valid period is presented as if it exist, with a default value of zero if it has never been edited. Internally, they are only stored once they have been edited.

For UploadBatches, only create actions are defined, because they represent a record of an upload happening, which is not something that can later be changed.

Entity			
Customer	Create	Change	Delete
BudgetProposal		Change	
Turnover		Change	
SalesPerson	Create	Change	Delete
Area	Create	Change	Delete
ProposalYear		Change	
UploadBatch	Create		

Table 3: Entities and the actions that can be performed on them

Along with the actual event store, this module contains a mechanism to construct an object-representation of (parts of) the current state. This is used to build the complete new state as result of an incoming change in order to send it out as a new event again. Additionally, where possible, the state-changes are specified in terms of "delta's" rather than overrides. For example for monetary values, this means that the difference in value is included in the event, instead of the new total value. On the other hand, if the name of a Customer changes, this cannot be expressed as a "delta" and the new name is simply sent as an override of the previous value.

The output of this model consists of events containing the complete state-change as result of each of the received inputs. However, if the state is not changed, no outgoing event is sent, because nothing noteworthy has happened.

Each type of event that is received by the EventStore module has a corresponding type of event that is sent out by the EventStore module as a result of it. For example, when a user changes a customer's data, a "CustomerChangeInputted" event is sent. This event is processed by the EventStore module and if the incoming event really results in a state change, a "CustomerChanged" event is sent by it. These two events seem similar, but their meaning differs. The "-Inputted" events carry the meaning of an external action; a user or external system (e.g. the SAP system) inputs some information about an entity

in the system. On the other hand, the events that are generated by the EventStore module mean that the state of the system has actually undergone a change.

7.4.2. SAPImport module

This module reads in XML files that are produced by the SAP system and translates them to events. These XML files contain new data that serves as input for the EDA-system.

Currently, the data contained in the XML files that needs to be entered into the EDA-system, is a set of turnover values. To keep a record of this process, a set of UploadBatches are used. These are sent in separate events at the beginning of the import process and are received and stored in a straight forward way by the EventStore module.

The main structure of the XML files is as follows. The entries are on top level separated by date and category. Below that, there is a set of salespersons (number + name), which each contain a set of customers (number + name). For each customer, there is a single turnover value.

This first of all means that the data needs to be turned inside-out. In the EDA-system, turnovers are first sorted by customer and within a certain customer, they are separated by date and category.

Secondly, SAP does not know the identifiers that are used within the EDA-system, so these need to be inferred by using the available data. To accomplish this, data in the XML files representing incomplete EDA-system customers and salespersons, are first put into separate incomplete entities. These are denoted by the prefix "Unknown", since the EDA-system does not know yet which specific entity is meant.

The subsequent import of a SalesPerson, one of its Customers and a Turnover for this customer is shown in figure 7. When the XML file references a specific EDA-system SalesPerson, the SAPImport module creates an UnknownSalesPerson with a newly generated identifier, and put its number and name into it. This UnknownSalesPerson is included in an event and sent. The EventStore module then receives this and matches it to an actual SalesPerson that already exists within the system. When the SAPImport subsequently sends an UnknownCustomer that was contained in that (Unknown)SalesPerson, it includes the identifier that it gave to that UnknownSalesPerson. The EventStore then deduces the actual SalesPerson for the Customer from that identifier it previously matched. The UnknownCustomer is matched to an actual existing Customer by using the SalesPerson it belongs to and its own number and name. When the SAPImport sends the turnover data belonging to that Customer, it similarly uses the identifier of

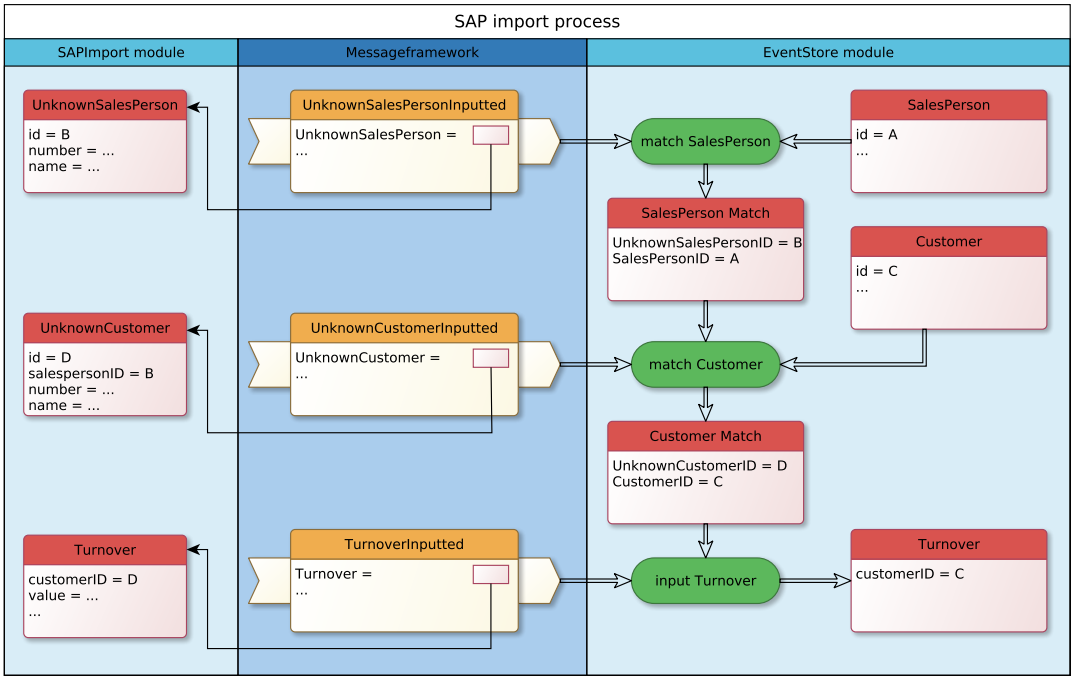

Figure 7: EDA-system – schematic representation of part of the SAP import process

the UnknownCustomer to link it to the actual Customer and enter the turnover data as a Turnover object belonging to that Customer.

7.4.3. Gateway module

The Gateway module contains the user interface and handles all user interaction. In the original system, this was done using an MVC library. However, for the EDA-system, this is not possible. The MVC library runs the web application on an ASP.NET server. This server is not compatible with the used Messageframework, so the library could not be used.

As an additional requirement, to accomplish the event-driven characteristic of the new system, the user interface should be able to react and change upon changes in the internal state of the system. To accomplish this, the SignalR library was used and the application is hosted locally using Owin. The SignalR library enables method calls between the C# code on the server and JavaScript code on the client. This way, if a user is viewing a page, it can be updated in response to an event.

The part of the Gateway module that actually uses the SignalR library to deal with the connection between client and server, is contained in a separate .dll. This .dll is

referenced by the main part of the module. The main part contains the connection to the Messageframework. It receives events from the View-model modules (see section 7.4.4) and also handles requests from the client. For this purpose, it saves data from events so that this data can be provided to a client that requests it. Additionally, when the clients submit data that needs to be passed into the system, an event is generated and sent.

7.4.4. ViewModel modules

Each view module collects the data for one specific type of report that can be viewed by the user. This data is put into the desired format and an event is generated whenever a change occurs. These events are used by the Gateway module to provide the data to clients. All view-model modules are completely independent of each other. For each type of desired view-model, a new module can be constructed and added to the system.

7.5. Design notes

Some relevant choices, considerations and observations made during the design process are described in the following sections.

7.5.1. Top-level design change

There is a single difference between the initial and final top-level designs. This difference lies in the communication between the view-model modules and the front-end. Initially, the design was to have the view-model modules save their data in the form of JSON files. Whenever a client requests a piece of data, it is extracted from the corresponding JSON file by the front-end. This way, the view-data would also be persistent.

In the final design, this was changed to a communication through events. This means that the current state of the view-models while the system is running is duplicated an extra time, because it is now kept in both the respective view-model module, as well as the Gateway module.

The main reason for the change was the fact that the persistence of the view-data is not desirable. Whenever a view-model module is restarted, this data should be reconstructed from the current state of the EventStore module. Otherwise, inconsistencies can arise. This means it is better to keep the view-data as internal state of the view-model modules, rather than save it persistently. Additionally, this change also meant that the front-end

would become a real module because it communicates via events with the rest of the system.

7.5.2. Versioning system

The system used a versioning system, which is used for two purposes. One is the determination of event order and the other is conflict resolution through expected versions. Both are explained in the next paragraphs.

Event order

In the EDA pattern, there are no guarantees on the order of arrival of events. The events that are received by a certain module are not guaranteed to arrive in the same order as they were sent.

This property has different consequences for different modules, depending on the way they keep their internal state. The EventStore module keeps a record of all events it receives. Because of that, out-of-order arrivals can be corrected when building the state from the event store. Consequently, the module can treat create-events and change-events differently.

On the other hand, there are modules that internally just keep some state that they use for other actions. A change-event for a particular entity could arrive before its create-event. Because of this, each change-event needs to be handled in a way that, if the entity did not exist yet in the internal state of the module, it could be created in that instance. When created this way, only the data from the change is incorporated and the data from any previous events is added retroactively (as soon as it is available). Therefore, the difference between create- and change-events effectively disappears and they are treated the same.

Additionally, a delete event could potentially arrive before the create event of the same entity. This means that as part of the state of a module of this type, deletions need to be remembered, so that their result can be executed at a later time.

Expected versions

In the EDA-system there are two kinds of external input events: those *with* an expected version and those *without*.

Expected versions are a mechanism to solve a specific type of conflict. They are needed when updates on an entity are done based on the current version of this entity. For example, a user who wants to update customer details needs to see the current customer

details and then change the desired fields. The conflict occurs when the customer details change between the time the user views them and the time their update arrives in the rest of the system. In this case the update is based on outdated information. Instead of accepting the update and applying it to the current state, it is desirable to dismiss this update and inform the user. To diagnose cases where an update is based on outdated information, each state contains a version. When an update that relies on an earlier state is submitted, the version of this previous state is sent with the update as an expected version of the current state. These are therefore input events *with* expected version. In the EDA-system these input events are used when the input is done by a user.

Input *without* an expected version is done when the update is independent of the previous state. This type of update in the EDA-system is used for the updates from SAP, since the data from the XML files is not based on the state inside the EDA-system.

A situation of conflict resolution by using expected versions is depicted in figure 8. The red entries stand for some form of state, e.g. the customer information of a customer in the system. The green entries are actions. Both users have previously received the current state from the data storage module. They both update it and submit this update. The submit that is received first by the storage module gets processed without problem. The expected version is checked against the current version and they match, so the update is incorporated in the new current state. When the update from the other user arrives, the version check fails, because the expected current version does not match the real current version. The update is not incorporated and a notification is sent to make the user aware of this. The user will also have received the updated state, so if the changes they wanted to make are still correct, they can enter and submit them again.

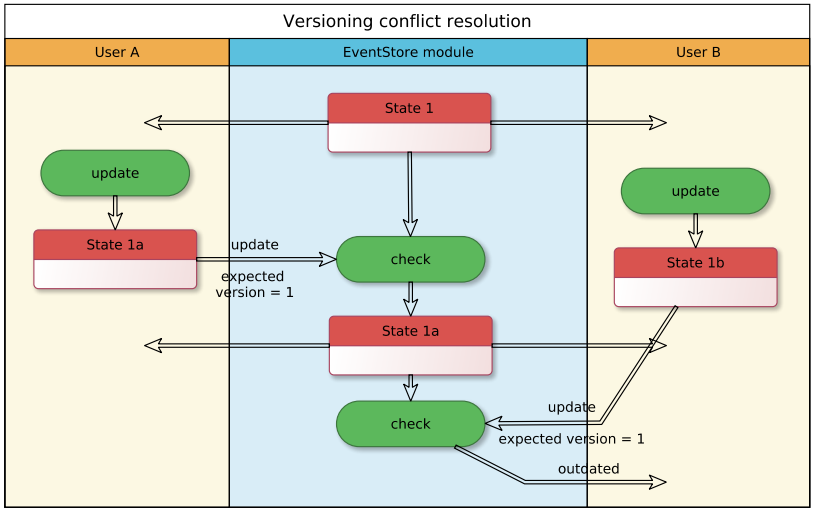

Figure 8: EDA-system – example of a versioning conflict

Since in the event-driven system changes in state are propagated to the clients, the depicted conflict only arises if the actions of both users happen within a small time window. However, since no guarantees can be made about how long it takes for events to reach their destination and therefore how small the time window is, this situation still needs to be accounted for.

Versioning system approaches

The versioning system that was used in the EDA-system was changed a few times due to new insights gained during the development process. At first, the version of an event was derived from the lastUpdate property of the entity that was included in it. This property took the form of a simple timestamp: the time that the last change to the entity was made, with a precision of milliseconds. This proved to be insufficient.

One of the problems was that the situation where two events received the same timestamp because of millisecond rounding was found to be practically possible, and occurred specifically during the SAP import process when a large amount of updates were send in rapid succession. This meant that the order of these events with the same timestamp could not be determined, which lead to arbitrary orderings and non-determinism, which caused problems in the system. To solve this, the version was extended to, aside from the timestamp, include a version number to solve collisions on the timestamp.

7.5.3. Entity-Relationship differences

The entity relationships of the original system are shown in figure 9. In this schematic representation, the red connection diamonds represent foreign key database table connection, while the orange ones represent a connection based on name (not enforced by the database). The green blocks represent enumerations, which do not have their own database table, but are referenced by name consistently throughout the other database tables.

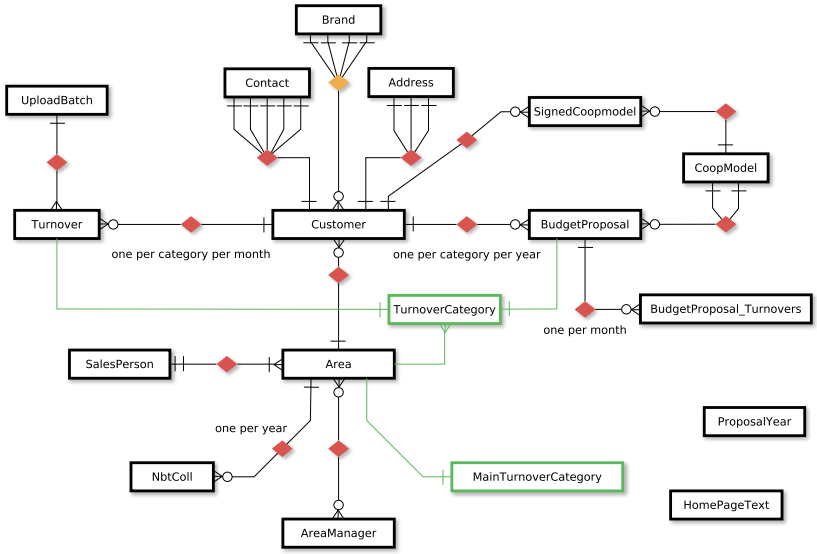

Figure 9: Original system – entity relationships

In the implementation of the EDA-system, the system was scaled down to scope the research project. As mentioned in section 7.4.1, some entities that occur in the original system are excluded as a result of this. To make a proper comparison, the original system was also scaled down in a similar way. The entity relationships in this reduced original system are shown in figure 10.

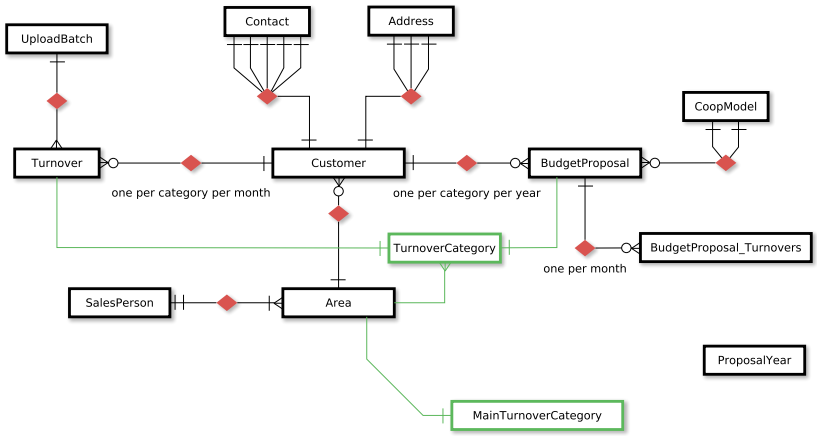

Figure 10: Original system – scaled down version of the entity relationships

In the EDA-system, the entities are defined for each module separately. Some modules may use the same representation, but this is not required. The representation of each entity within events sent between modules is knowledge shared throughout the system, but the representation within each module is only known to that module itself. The

entity relationships for the EDA-system discussed in this section are those used in the EventStore module, since this is where the domain model is located. The entities used in the communication between modules is virtually the same, with a few small differences, caused by limitations of the Messageframework.

The entities and their connections in the EDA-system are shown in figure 11. Because the database used in this system is not a relational database but an event store, the connection diamonds have slightly different meanings.

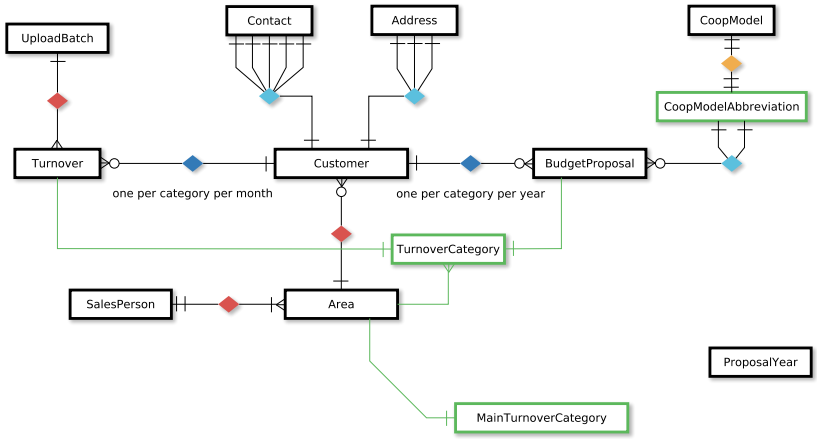

Figure 11: EDA-system – entity relationships in the (scaled down) implementation

The red connection diamonds represent entities that are stored in separate streams in the event store, but contain a link via identifiers. This means the entity in one of the streams contains a property that contains the identifier of the associated entity in the other stream. This is similar to a foreign key connection, but is not enforced by the database in any way.

The dark blue connection diamonds represent a connection between objects that are stored in the same stream. Between them, there is a "has a" relationship. For example, the relationship between Customer and BudgetProposal is that each BudgetProposal is linked to one customer. This means that the relationship is more asymmetric, because the contained entities (in the example this would be the BudgetProposal) exist in the stream of the containing entity and are therefore dependent on it.

The light blue connection diamonds represent an even more dependent relationship. Here the contained entity is just a property of the containing entity. For example, the relationship between Customer and Address is that each Customer has an Address (or in this case three addresses, which could be the same). Here, the contained entity is not considered a separate entity any more and is, in the top-level design, treated as any other property of the containing entity. They are still displayed as separate entities in

the figure to show the differences with the original system.

The orange connection diamond is again a connection based on name and the green blocks represent enumerations, which only occur in the database as a named property, but are consistently used throughout the domain.

The main difference between the two systems is that the original system treats every piece of data that has its own class as a separate entity in the domain model. These entities are all stored in separate database tables and connected by their keys. This has the advantage of being able to put restrictions on the entities, which is enforced by the database. On the other hand, it does mean that whenever the data is required, extra actions are needed to get all required data together. In the EDA-system, only some entities are stored completely separately. Others are stored within the same event store stream. This makes it easier to gather all necessary data, but the event store does not enforce any restrictions, so wherever needed, this needs to be explicitly done within the system.

8. Case study: Measurement methods and results

In this section, the measurement methods are described and results of these measurements are presented. The used measurement methods are already briefly mentioned in section 4.4, but they are described in more detail in this section. The results of large measurements are summarized; the corresponding complete numeric data can be found appendix B. An interpretation of the results and comparisons between both systems is given in section 9.

The changes in the change-scenarios were actual changes in the requirements of the system. Therefore, the code metrics as well as the time and resource measurements were conducted on the post-change-scenario systems, as these are the forms closest to the system desired by the client. All measurements on the original system were carried out on the reduced version of it, to make the results comparable to the results from the EDA-system, which does not have an implementation of the whole system.

The reduced original system and the EDA-system provide almost the same amount of functionality, with one notable difference. The original system also includes account functionality, i.e. users that can log in and roles with certain permissions. It would have been desirable to remove this from the reduced form of the original system, but within the ASP.NET design, this was very much woven into the core of the design and therefore not trivial to remove. The choice was made to leave it in, because the

impact on measurement results was expected to be small and time constraints meant that spending effort on this meant less effort could be spent on other aspects. The alternative of adding this functionality to the EDA-system would require even more effort because of the use of a different kind of web server for which this functionality was less integrated. This difference between the two systems is expected to have little influence on the measurement results, but it should still be considered when comparing the results. This is done explicitly in section 9 for quality aspects where it is expected to be of significant importance.

8.1. Change-scenarios

As mentioned in section 4.4.4, a small set of change-scenarios is used to evaluate the Modifiability and Extensibility of the two systems. For the scenarios, actual changes that have been made in the original system in reaction to changes in the requirements, are used. The used change-scenarios are first described and then the amount of effort they required in both systems is evaluated. This amount of effort is expressed in the total amount of added and removed lines, the amount of adapted classes and the amount of modified development units (namespaces and modules). The amount of effort for the original system is gathered from the SVN repository and is adjusted to correspond with what would have been changed in the reduced version of the system.

8.1.1. Scenario 1

The first change-scenario consists of a change in the information that is displayed in one of the reports. This report shows some specific information about each Customer in a selected area. It is presented in a table where each column specifies a piece of customer information that is displayed in the report. In the original system, this report is generated on demand, it does not pass through the batchpdfcreator. Initially, two of the columns in the report contained data combined from the BudgetProposals of the Customer. These columns were no longer necessary in this report and instead, telephone numbers needed to be displayed.

In the original system, this meant a change in the MVC part of the system. One Controller, ViewModel and View needed to be slightly changed. Additionally, a function in one of the Services was no longer used after the change. It had not been removed in the actual system, but here it is still counted as being removed. This is done for consistency between the systems.

In the EDA-system, the change was made by reforming the corresponding view-model module. Because this specific change meant the information was much easier to gather,

there were quite a few deletions involved. Specifically, the previous ViewModel needed information from BurgetProposals, but the new one no longer did. Therefore, all code involving the receiving and processing of BudgetProposal events could be removed. In addition, a small change in a front-end JavaScript file was required in order to display the new report correctly.

Table 4 shows the required effort for both systems. The complete data can be found in tables 19 and 22.

	modules	namespaces	classes	additions	deletions
Original system (reduced)	2	4	4	22	30
EDA-system	2	2	7	15	442

Table 4: Results of change-scenario 1

8.1.2. Scenario 2

The second change-scenario involves a change in the way customers are handled. Originally, customers where completely independent from one another. However, it occurred that several customers existed that were actually different departments of the same customer. These were split up because they were handled by different SalesPersons in different Areas. This meant that the basic information about these customers had to be managed in several different places and kept consistent manually. The desired situation was one where a set of common fields was defined that would be the same for a set of these. To achieve this, the concept of customer 'linking' was introduced, where linked customers would share part of their information. The change-scenario is actually split into two parts. The first is the domain change to allow for and handle linked customers. The second is the functionality that allows for the act of creating a linked customer. A third thing that needed to happen in practice, but is left out of the change-scenario, is the reformation of the existing data in the system and adding links between customers where needed.

In the original system, this was achieved by changing the Customer entity to mean the collective of linked customers. For the information that was not shared between them, the concept AreaCustomer was introduced. What previously was a Customer, was now uniquely defined by the combination of a Customer and an Area. This meant that the database structure needed to be reformed in addition to the source code. The new entity relationship diagram for this situation (still in the scaled down form) is shown in figure 12. Just as in figure 10, the red connection diamonds represent foreign key connections and the green boxes are enumerations, which do not have their own database table.

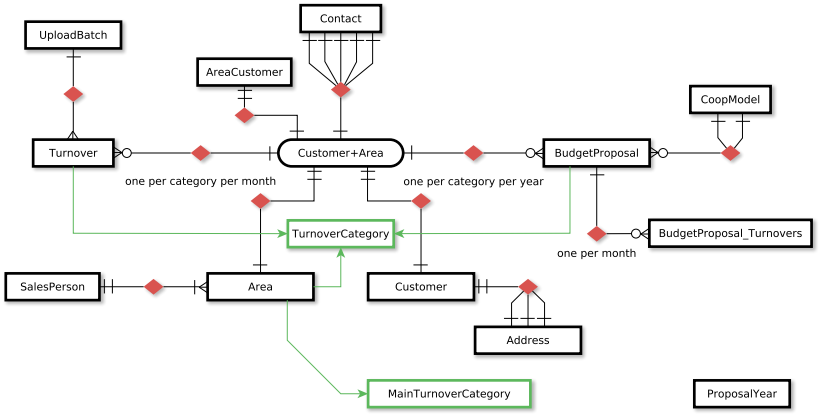

Figure 12: Original system – entity relationships after the customer link change-scenario

In the EDA-system, it was achieved in a different way. The concept of Customer was kept the same and the link between them was achieved by adding a linking event to the respective event store streams. Then, when one of the streams receives an update on a shared field, the update is applied to all other customer streams that it is linked to as well. This required some modifications to the event store storage conventions to make sure expected version checking would still behave as desired.

Table 5 shows the required effort for both systems. An additional entry marked (*) is added here, because even though some of the required changes fell outside of the scope of the reduced system, they correspond with features that, would they have been added into the EDA-system implementation, they would almost certainly not require additional changes in this change-scenario. What this table does not show is that, in addition to modifications to source code, the original system also required the database structure to be changed for this scenario. The complete data can be found in tables 21, 20 and 23.

	modules	namespaces	classes	additions	deletions
Original system (reduced)	4	6	15	1035	310
Original system (*)	4	8	23	1114	371
EDA-system	3	4	16	367	90

Table 5: Results of change-scenario 2

8.2. Code metrics

As mentioned in section 4.4.2, for both systems a few code metrics were determined, using Visual Studio Code Metrics Powertool and Code Metrics Viewer. The tool calculates the metrics based on an intermediate representation made by the compiler, so they do not

always correspond directly to the source code. It can determine the metrics on various scopes from module-scope to method-scope. For each metric, the scope or scopes are chosen where the metric gives useful information with respect to the chosen quality attributes. The tool itself also gives some interpretation in the form of two thresholds per metric. Values within the first threshold have a good score ("green"), those between the thresholds have a moderate score ("orange") and those outside the second threshold have a low score ("red").

The results are given in a frequency-based way. This section mostly includes histograms to visualize the data, while appendix B contains complete numeric data.

8.2.1. Lines of Code

This metric is fairly simple. It counts the number of lines of program source code. As mentioned earlier, the tool used some intermediate representation, so the line counts do not correspond directly to the actual number of lines in the source code. For example, comments and line-breaks within a statement are not counted as extra lines.

For this project, only the total number of lines for both systems are included, because information about the way the lines are distributed is not useful with respect to the aspect of Conciseness, for which this metric is used. The results are shown in table 24.

	Lines of Code
original system	3787
EDA-system	3974

Table 6: Lines of Code metric results

8.2.2. Cyclomatic Complexity

This metric is a measure for the number of decision points. The metric makes most sense to measure on the method-scope, since there it gives an idea about the complexity of a set of a unit of instructions. A higher value means more decision points, so higher complexity, which means a lower score. The thresholds given by the tool are at the values of 10 and 20.

The results are shown as histograms in figures 13 and 14. Here, instead of showing the value for each individual rating, the ratings are added together over small ranges. The full results can be found in table 24.

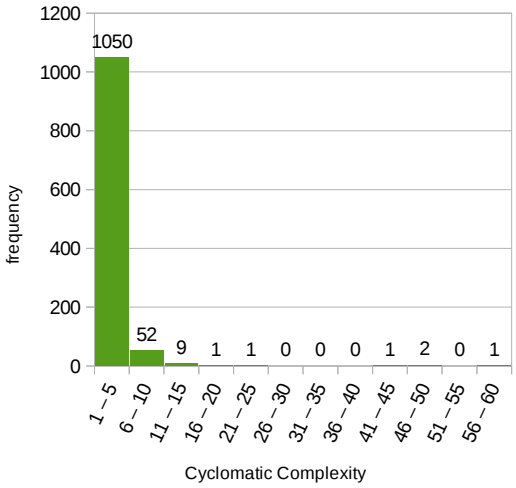

Figure 13: Original system – Cyclomatic Complexity

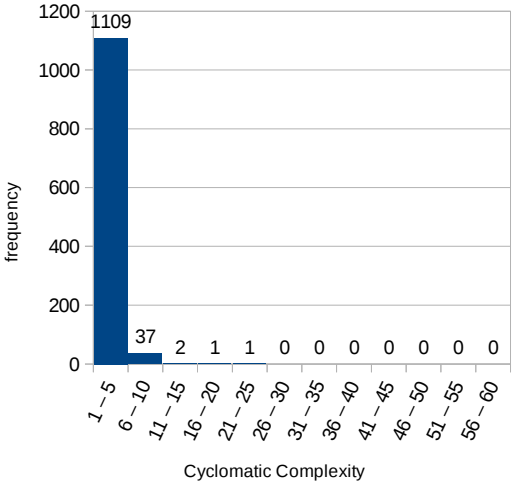

Figure 14: EDA-system – Cyclomatic Complexity

8.2.3. Class Coupling

This metric measures the amount of references of one class to other classes. A higher value means more dependency on other classes, which means a lower score. The thresholds given by the tool are at the values of 9 and 49.

The results are shown as histograms in figures 15 and 16. Just as with the Cyclomatic Complexity results, the frequencies for the ratings are added together over small ranges. The full results can be found in table 24.

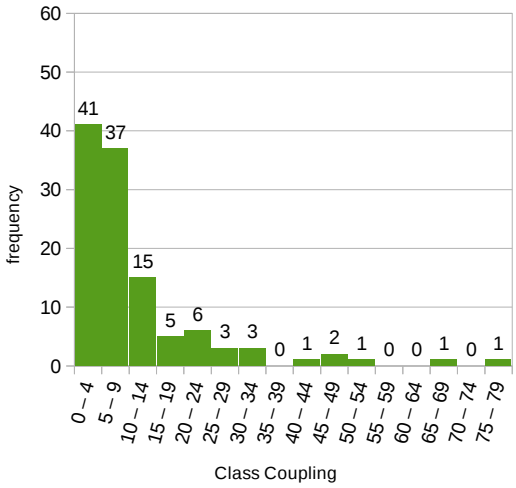

Figure 15: Original system – Class Coupling

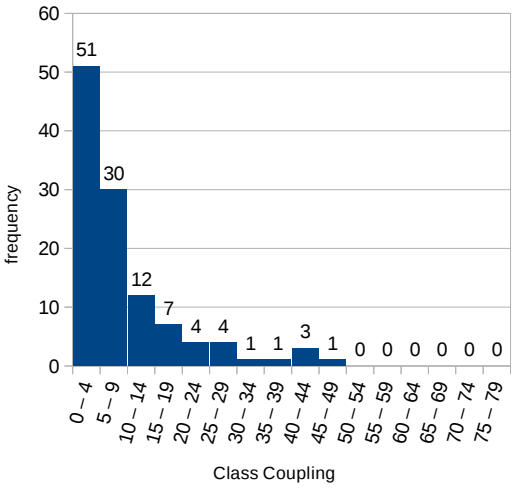

Figure 16: EDA-system – Class Coupling

8.2.4. Depth of Inheritance

This metric measures the depth of the inheritance tree of classes. A higher value means a class has more ancestors, which means a lower score. The thresholds given by the tool are at the values of 2 and 4.

The results are shown as histograms in figures 17 and 18. The full results can be found in table 24.

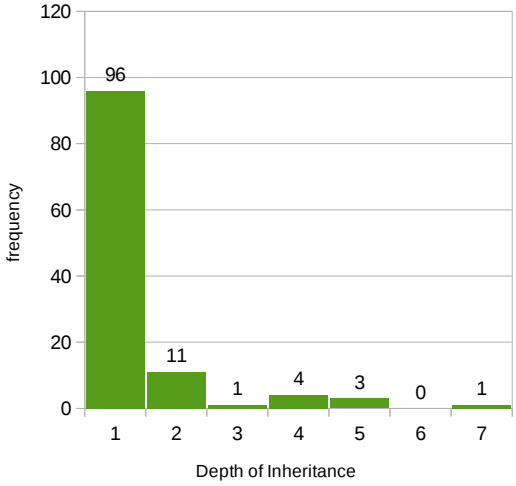

Figure 17: Original system – Depth of Inheritance

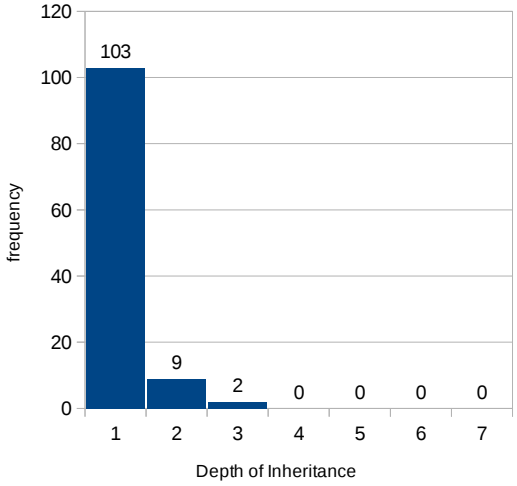

Figure 18: EDA-system – Depth of Inheritance

8.2.5. Maintainability Index

This metric gives the results of a formula, originally presented in [34]. The formula combines other metrics into one maintainability score. The formula used in the used tool is slightly modified. The scale is divided by ten and the optionally included metric about the percentage of lines of commentary in the original formula is removed. The formula has since received criticism. It is based solely on empirical fitting and takes the average of metrics like the Cyclomatic Complexity, for which the average does not give a good indication of the overall quality [35]. For these reasons, not too much importance should be placed upon this metric. It is still included here to give a rough indication of some maintainability aspects.

In contrast with the other metrics, here a higher value means better a better score. The metric is used to give a rough indication, so it is applied on module-scope. The thresholds given by the tool are at the values of 20 and 10, which is far below any of the measured values.

The results are shown in figure 19. AreaList and CIFIndexList are the two implemented ViewModel modules (see section 7.4.4).

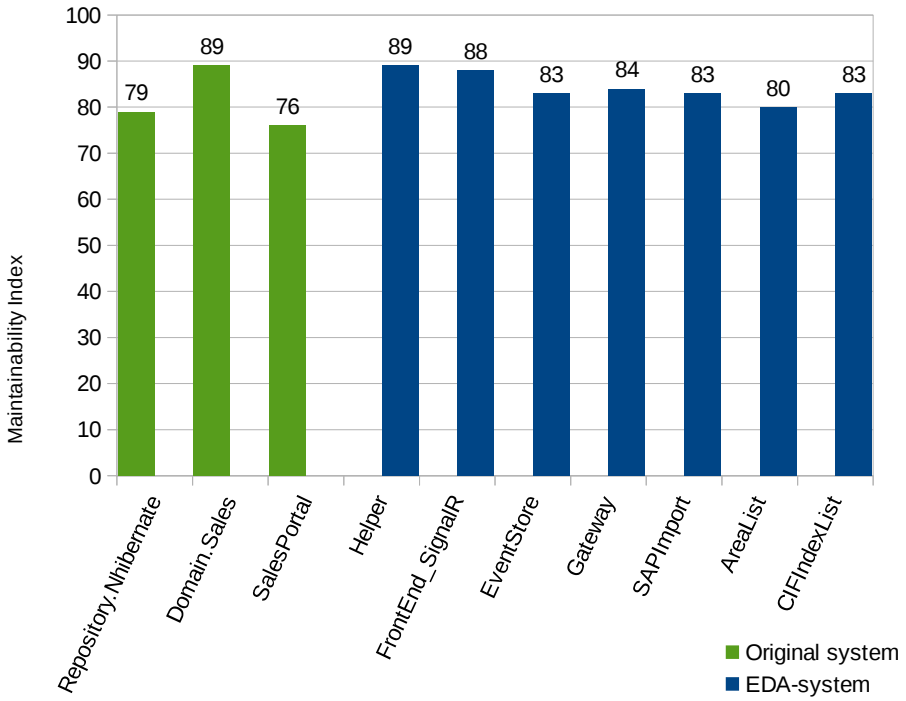

Figure 19: Maintainability Index

8.3. Time measurements

The main time measurement involves web page loading times. For both systems, the page showing the report mentioned in section 8.1.1 was loaded for a set of different areas. Each area contains a different number of customers, as shown in table 7. As previously mentioned, this report is generated on demand in the original system, while in the EDA-system, it is pre-generated like every other report. The original system does contain reports that are also pre-generated, but for the comparison it is more interesting to look at a report which highlights the differences between both systems.

area no	customers
100	907
20	175
63	120
34	66

Table 7: Areas and the number of customers they contain

As mentioned in section 4.4.1, the loading times were measured using a tool called HttpWatch. With this tool, the HTTP loading times were measured, as well as the completion times of requests to the server.

The measurements for both systems were conducted by running the system four times, each time taking twelve consecutive measurement point. Each measurement point measured the HTTP loading time of a page, as well as the server request completion time. The twelve measurements points were divided into four sets of three. Each set measured the loading time of the page for one specific area, and this measurement was repeated three times. For example, one run would be:

20, 20, 20, 100, 100, 100, 34, 34, 34, 63, 63, 63,

where each measurement point is indicated by its corresponding area number. For each run, the order in which the areas were measured was changed, to see if any aspects of the order influenced the measurements. The measurements of the same area are grouped together for practical purposes; in the original system, switching between areas requires logging out and back in to the system.

For each system, the results of these measurements are shown in two different graphs. The first shows the measurement points of each run in chronological order, while the second has the measurement points sorted by area. The complete numeric data can be found in tables 25, 26 and 27.

Original system

The two graphs of the results for the original system are shown in figure 20 and 21 respectively. Figure 20 shows both the HTTP load times and the server response times for each run. figure 21 shows only the server response times.

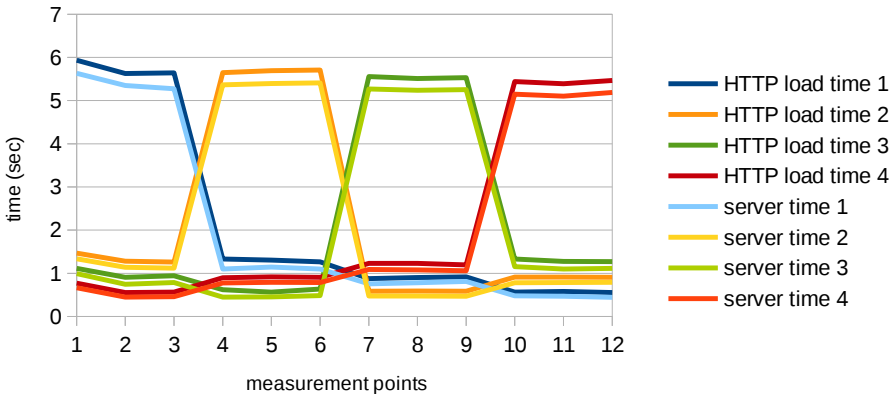

Figure 20: Original system – loading time measurements

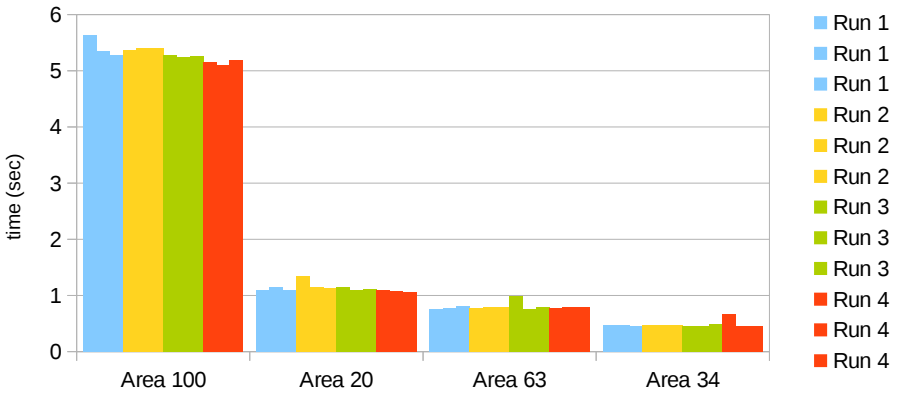

Figure 21: Original system – loading time measurements grouped by area

EDA-system

The two graphs of the results for the EDA-system are shown in figure 22 and 23 respectively. For the EDA-system, the HTTP load times were equal to the server response times. This is because the page building is done by JavaScript code, which lies outside of the measuring capabilities of HttpWatch.

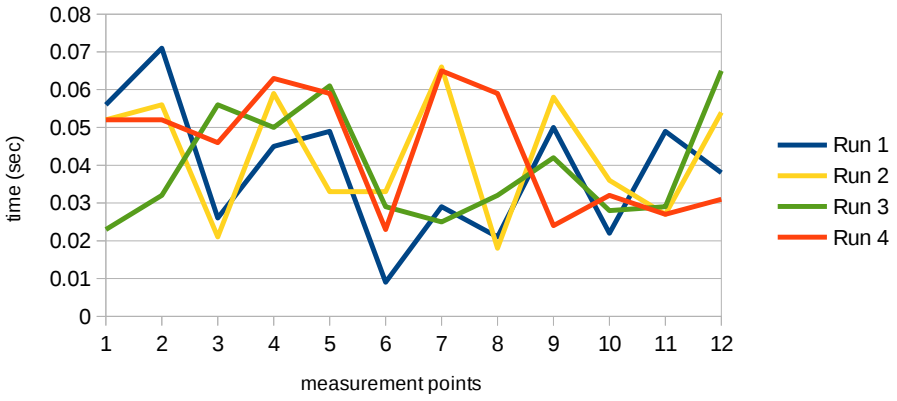

Figure 22: EDA-system – loading time measurements

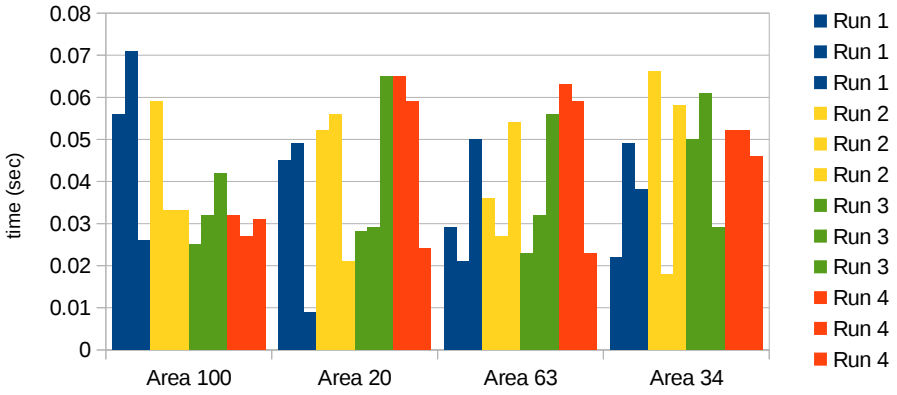

Figure 23: EDA-system – loading time measurements grouped by area

8.4. Resource measurements

Resource measurements are conducted on a few different aspects of the systems. First of all, the disk storage space of the databases is measured. The results are shown in table 8. As shown in the table, of the event store, only the occupied space on the disk is known, but not how much of it is actually used.

	database type	database size (MB)	used space (MB)
original system	relational database	673.69	225.38
EDA-system	event store	256	*unknown*

Table 8: Database sizes

Secondly, the amount of memory used during runtime is determined. To do this, two methods were used. The first is a rough measurement through the Windows Task Manager. The second is using C# functionality that counts the total memory in use at specific times during runtime. This functionality is temporarily incorporated within the source code of the two systems. This means this measurement method could slightly influence the behaviour of the system. However, no large differences were observed, so the effect is most likely negligible.

The second method is not possible for the extra processes that need to run besides the main program. For the original system, this is the SQL server and for the EDA-system, it is the EventStore server. The memory usage of these extra processes could therefore only be roughly determined. The results, combined with the rough measurements of the main processes (the web servers) of both systems, are shown in table 9.

system	process	average memory usage (K)
original system	SQL server	1.4×10^5
original system	web server	0.3×10^5
EDA-system	EventStore	2.4×10^5
EDA-system	web server	1.0×10^5

Table 9: Process memory usage from Windows Task Manager

For the main programs of both systems, the second approach was used to do more systematic measurements. First of all the base memory usage of both systems was measured after start-up of the systems. To simulate usage, the same set of pages that was used in the time measurements is used. The influence on the memory usage while loading these pages is monitored. The complete numeric data can be found in tables 28, 29 and 30.

Original system

For the original system, just as in the time measurements, the memory measurements were done in four different runs. There are two different kinds of measurement points: ground points and peak points. Ground points are points where the stable memory usage at that point during runtime is measured. Peak points are points where the memory usage is measured just after an intensive task, in this case the loading of a report. The runs consist of alternating ground and peak points, so that the increase in memory usage for a peak can be determined, as well as the effects of page loading on the ground memory usage. The pages for different areas are loaded in different orders for different runs to determine any effects of the order on the measurements.

Figure 24 shows the four runs, including peak points and figure 25 shows only the ground points for the four runs. In both figures, the x-axis represents the progression of measurement points; measurement point 1 is the first measurement, measurement point 2 is the measurement after that, etc.

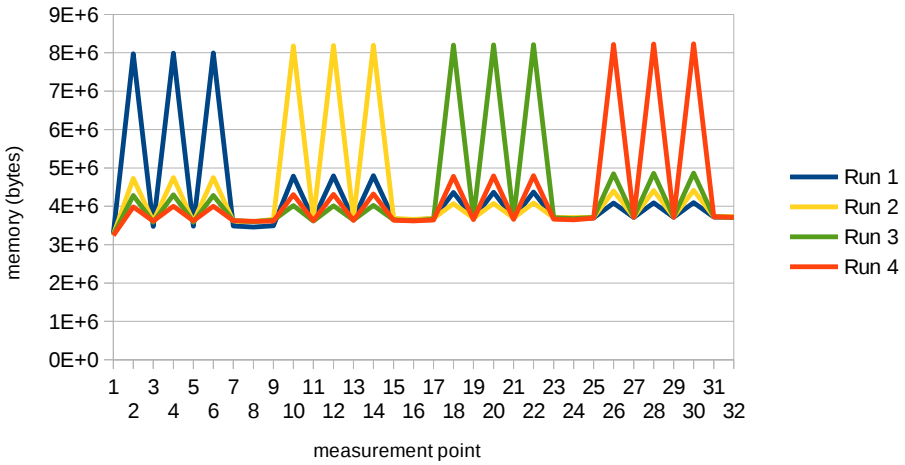

Figure 24: Original system – memory consumption measurements

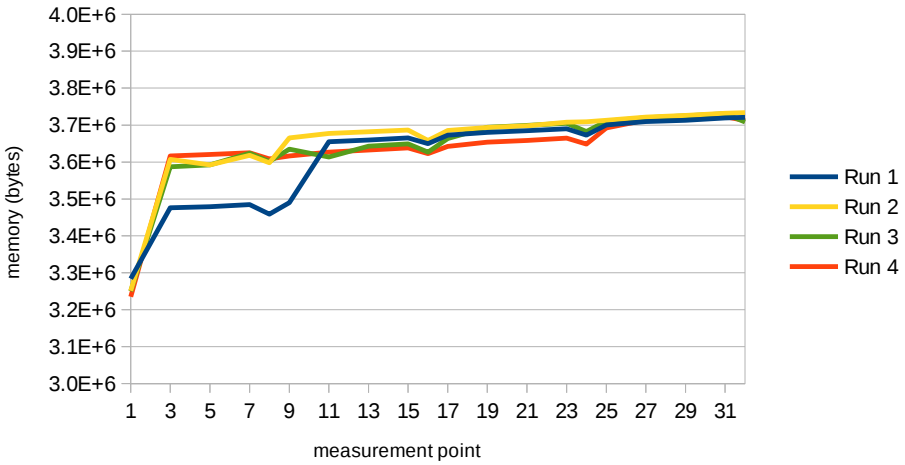

Figure 25: Original system – memory consumption measurements without the peaks

EDA-system

In the EDA-system, the order of the areas was also varied between measurement runs. Here, the measurements of the same area are not grouped because in the EDA-system, switching between areas is easier than measuring the same area consecutively.

No peaks are observed, so the results are shown in a single graph (see figure 26). Again, the x-axis represents the progression of measurement points. Since not all runs contain an equal amount of measurement points, the longer runs are truncated in this graph.

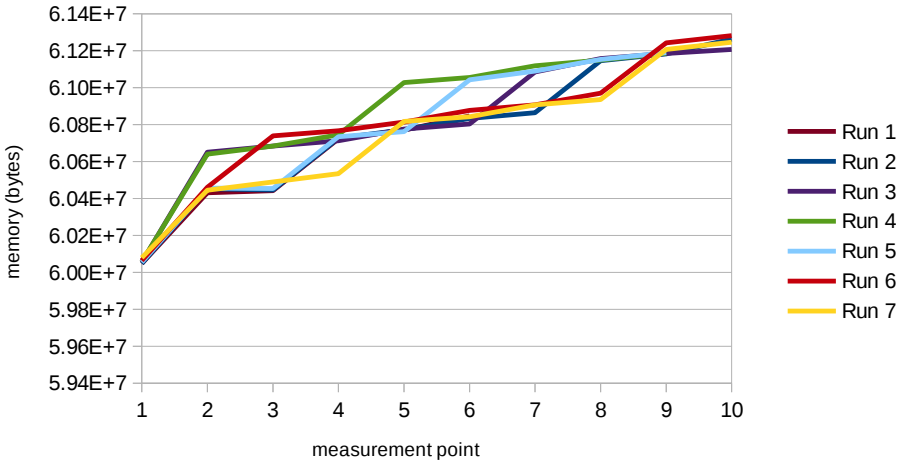

Figure 26: EDA-system – memory consumption measurements

9. Case study: Interpretation and comparison

In this section, the results presented in section 8 are interpreted and a comparison with regard to the quality criteria selected in section 4 is made between the two systems. The measurements from section 8 were all made on the implementations, which implement only part of the total architectures. Therefore, some generalizations are made where applicable, in order to make statements about the whole architectures. Also, according to figure 1, some quality aspects are dependent on other aspects. To avoid repetition, for aspects that depend on others, only additional measurements or insights are presented.

9.1. Efficiency

To compare the Efficiency characteristics of both systems, the data from time measurements (section 8.3) and resource measurements (section 8.4) are used.

9.1.1. Time behaviour

For the time measurements, the average loading time for each area is calculated for both systems, so they can be compared.

For the original system, figure 20 shows that the order of the areas does not matter, except for the fact that the very first measurement in a run is always a little higher than

other measurements for the same area. This can also be seen in figure 21, where each area has one slightly higher bar, which corresponds to the very first measurement in a run. This is probably because some extra actions are required the first time a report is constructed, for example regarding the connection to the database. To obtain accurate averages, any factor regarding the order of the measurements should be taken out where possible. Therefore, in calculation the averages, this first measurement of a run is left out. The results are shown in table 10.

area no	HTTP load time (sec)	server response time (sec)
100	5.563	5.272
20	1.270	1.109
63	0.908	0.781
34	0.581	0.463

Table 10: Original system – average loading times per area

For the EDA-system, figures 22 and 23 show no dependence on the order of the measurements, so all data points are used in calculating the averages. The results are shown in table 11.

area nr	server response time (sec)
100	0.039
20	0.042
63	0.039
34	0.045

Table 11: EDA-system – average loading times per area

In the comparison, for the original system, the server response time is taken, since the nature of the time measurements taken in the EDA-system correspond most closely to that measurement. When comparing tables 10 and 11, it is clear that the server response times of the EDA-system are significantly smaller than those of the original system. This was expected, since the original system makes the requested report at the time of the request, while the EDA-system collects and prepares all necessary data beforehand.

In addition to the measured page load times, other timing aspects of the system can also be considered. First of all, the long batchpdfcreator process that is run every night in the original system is no longer needed in the EDA-system.

On the other hand, the EDA-system takes a significantly longer time on start-up before all data is available throughout the system. In the current implementation, it takes

about 5:30 minutes, but this was not precisely measured. The explanation for the long start-up time is that on start-up, all available data needs to be extracted from the event store and pushed into the rest of the system by the EventStore module. The amount of time would be even larger than in the current implementation if the entire system would have been implemented and all modules are run at the same node. However, if the data pushing is not done too quickly (so that the modules have time in between to process other events), the system can already be used during this time, if users take into account that not all data may be available yet. Additionally, the system does not need to be restarted frequently. This only needs to happen when a failure occurs, or if some developmental changes to the system are put into production.

Finally, the SAP import process takes a bit longer in the EDA-system, because the data is immediately processed and distributed to all modules. This would again increase further if the entire system would have been implemented. Also, like during the start-up, the system can still be used while the import process takes place, if users take into account that incomplete data may be displayed meanwhile. In contrast, in the original system the SalesPortal is shut down during import (overnight).

9.1.2. Resource behaviour

First of all, the average starting memory usage of both systems is calculated from the measurements and shown in table 12.

system	starting memory usage (bytes)
original system	3255110
EDA-system	60431568

Table 12: Average starting memory usage

To analyse the additional memory measurement results, two aspects were analysed. These were the memory peaks (which only occurred in the original system) and the increase in ground level memory usage as result of page loading.

The original system contains peaks in memory usage whenever a report page is loaded. This means that during the construction of the report, extra memory is required by the system, which is released again after the report construction is finished. To determine how much extra memory is required for each report, the height of the peaks relative to the ground memory usage is calculated. The results are displayed in figure 27. This figure shows five outliers, four of which correspond to the first measurement in each run. The fifth corresponds to the first measurement of area 20 in run 1. In all these outlying

points, some additional actions were most likely performed that caused the system to use extra memory. Therefore, these points are not included when calculating the average values shown in table 13.

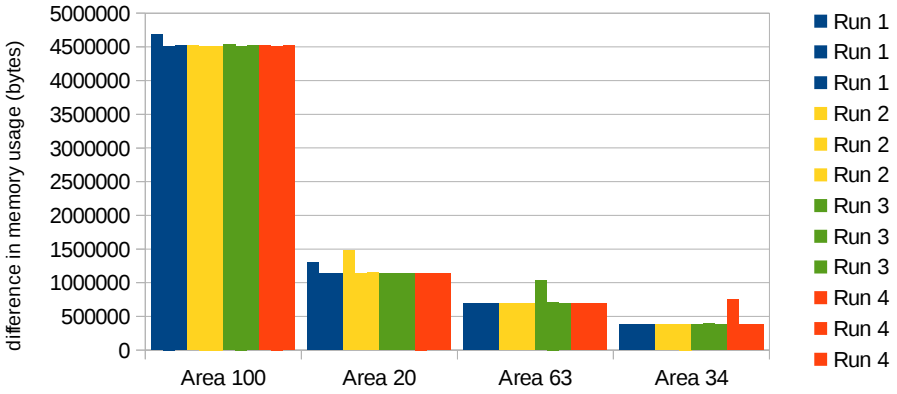

Figure 27: Original system – peak memory usage relative to ground memory usage

area nr	peak memory usage (bytes)
100	4516945
20	1140054
63	690803
34	384605

Table 13: Original system – average peak memory usage

The increase in ground level memory usage for the original system is calculated by subtracting the ground memory usage before a peak from the ground memory usage after the peak. The results are shown in figure 28, where the same outliers as in the peak analysis are observed. Additionally, the figure shows no real correlation between area and increase in ground memory usage. This means all dependence on the area (and therefore on the number of customers) is in the peak memory usage and not in the ground level increase. The average ground memory usage increase is therefore not calculated per area, but for the complete set of measurements as a whole. In this calculation, the outliers are again left out. The resulting average increase in ground memory usage is **7952** bytes.

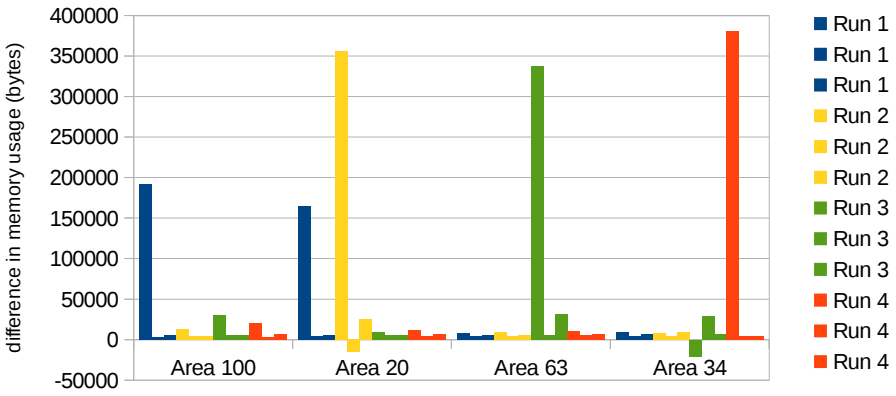

Figure 28: Original system – increase in ground memory usage

The EDA-system contains no peaks, so only the differences in ground memory usage are determined. Since the runs for these memory measurements were not all equally long, the data are more clearly displayed as lines instead of bars (see figure 29). As seen in the figure, the increase in memory is dependent on area and the noise level in the measurement is relatively high. To still get representative averages, a larger number of measurement points were gathered, hence the longer measurement runs. The calculated averages are shown in table 14.

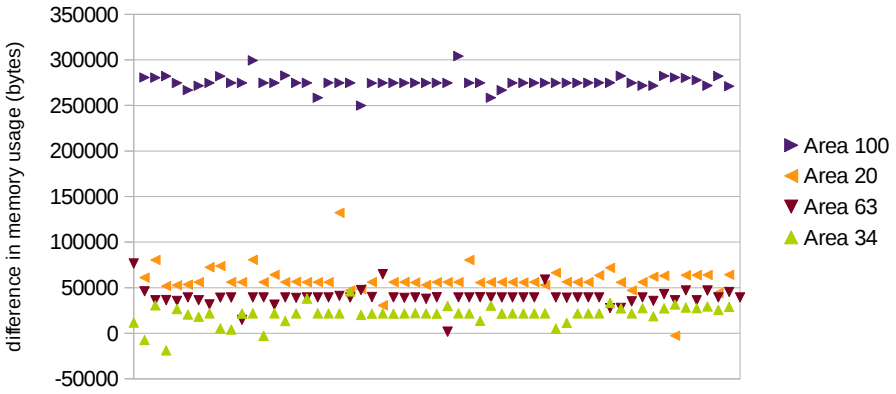

Figure 29: EDA-system – increase in ground level memory usage

area nr	increase in memory usage (bytes)
100	275438
20	58543
63	39246
34	20815

Table 14: EDA-system – average ground level memory usage increases

When comparing the resource behaviour data from both systems, a few different things can be noted. First of all, table 12 shows that the starting memory usage of the original system is significantly smaller than that of the EDA-system. This was expected, since the EDA-system prepares all reports beforehand and keeps them in memory. This difference would be even larger if the entire system would have been implemented.

Additionally, only the original system showed peaks in memory usage when loading report pages. This is as expected, since the original system creates the reports on page request and needs extra memory to do so. This memory is no longer needed once the report is completed. It is also as expected that this extra peak memory is dependent on the area and therefore on the number of customers contained in the report, because more customers means more data necessary to create the report. Since all this extra memory is released afterwards, it is not surprising that the increase in ground memory usage in the original system is relatively small and independent of area. This memory increase is probably the result of some sort of recollection of data exchange between the system and the database and/or between the server and the client.

In the EDA-system, the increase in (ground) memory usage *is* dependent on the area. This is probably the result of communication between server and client. In some runs, the memory usage suddenly dropped back down to the starting level, so the extra memory usage seems to be temporary. It could be a temporary storage of transferred data between client and server. The fact that the same thing is not observed in the original system could have multiple explanations. The first is the fact that in the original system, the web server part is not explicitly included in the system, but is managed by ASP.NET, so its memory usage may not be included in the scope of the *C#* measurements. Alternatively, the different used web servers and ways of client-server communication could have different memory usage conventions.

On a larger scope, tables 9 and 12 are combined to show the difference in results from the two different memory measurement methods (see table 15). The table shows that the memory measurements from both methods differ by a large amount. This is probably because some extra memory is being used by the running processes that is outside of the measuring scope of the C# functionality. This extra memory for both systems is of

comparable size, especially given the low precision in the measurement from Windows Task Manager (WTM) and therefore also in the calculated differences.

system	memory from WTM (K)	memory from C# (K)	difference (K)
original system	0.3×10^5	0.03×10^5	0.3×10^5
EDA-system	1.0×10^5	0.60×10^5	0.4×10^5

Table 15: Memory usage from different measurement methods

Table 9 also shows the memory required by the extra processes for both systems. The EventStore process used in the EDA-system uses a significantly larger amount of memory, but the two are still within the same order of magnitude. Furthermore, all process memory usages are still small compared to the memory usage of for example a standard web browser like Firefox or Chrome. This means that the differences, although significant, is usually of little consequence on modern machines.

The same holds for the disk storage spaces from table 8. The total disk space used by the SQL database from the original system is significantly larger than the space used by the EventStore, but both are of very manageable size for modern machines.

9.2. Structuredness and Understandability

For both Structuredness and Understandability aspects, the data from Code Metrics (section 8.2) are used. Additionally, observations are used for a qualitative comparison.

9.2.1. Conciseness

The Conciseness of both systems is mainly determined by simply looking at their relative sizes. The results of the Lines of Code metric are used for this. When comparing the results from table 6, only a small difference is observed, where the EDA-system uses a little more lines.

A few notes need to be made regarding this comparison. First of all, as mentioned in section 8, the original system provides a little more functionality. Therefore, the Lines of Code value for the original system without this extra functionality would be a little lower.

On the other hand, in the EDA-system many operations regarding the EventStore are done manually. The need for part of these operations could have been avoided by using more of the EventStore API. However, without spending a considerable amount of

effort, the information on which functionality was provided and how to use it could not be extracted from the documentation of the used EventStore. Because of the limited scope of the research project, this was not followed up on, but this decision considerably inflates the size of the EDA-system.

All things considered, the difference in Conciseness between the two systems is relatively very small, so both systems are about equally concise.

9.2.2. Consistency

For Consistency, no quantitative measurement methods were found, so a qualitative comparison is given instead. The low-level Consistency regarding naming and notational conventions is independent of higher-level design choices and is therefore not considered.

For high-level Consistency, there exists a difference between the modules in the way in which reports are generated. In the EDA-system, the information for each report is collected by a separate module. The information is updated whenever new information is available. The collected information is stored on the server and delivered to the user on request. This means for all reports, the same method is used. In the original system, some reports are generated on user request by a Controller. Others are created in batch overnight and are included as HTML tables in the web application. On this point, the EDA-system is more consistent.

9.2.3. Modularity

To determine the Modularity of both systems, the results from the code metric Class Coupling are used. Since it is an Object-Oriented metric, it measures modularity on a lower level than the level on which modules in EDA are specified. Nonetheless, it can measure the modularity within the top-level modules, which is influenced by the coupling and cohesion on the higher level. It is therefore still relatively useful with respect to the comparison.

Figures 15 and 16 show that a power distribution (as is used to analyse some of the upcoming metrics results) does not quite fit here. This is expected, because a little coupling between classes should be quite common, especially because the tool includes classes from system libraries in the Class Coupling count.

A qualitative comparison between figures 15 and 16 shows that the Class Coupling in the EDA-system is overall a little lower and has fewer high values. This indicates that the Modularity within the EDA-system is a little better than in the original system, but there exists no large difference.

One of the advantages of the EDA-system is that there is per definition no Class Coupling between classes in different modules. In the original system, this is not the case, so coupling between top-level modules is not regulated and can occur throughout the classes within these modules. In the EDA-system, the coupling is in the form of events, which are sent and received at specific spots within the modules and can therefore be monitored more easily. This makes the EDA-system more modular on the top-level of the system.

9.2.4. Simplicity

To compare Simplicity between the two systems, the code metrics Cyclomatic Complexity and Depth of Inheritance are used. Cyclomatic Complexity measures a form of code complexity, which is useful since Simplicity is the absence of complexity. Depth of Inheritance gives the amount of ancestors of a class. If a class has more ancestors, it is more difficult to track all available and exposed methods and it is less clear how the class functions.

The Cyclomatic Complexity is a low-level metric, but it measures the effect of the higher-level design choices and is therefore relatively useful in the comparison. The Depth of Inheritance is a very Object-Oriented metric, and its results can be less contributed to a difference in high-level architecture between the systems. They still give some indication of overall complexity and are therefore still included, but the results are not weighed heavily in the comparison.

The results from the Cyclomatic Complexity are expected to follow a power law [35]. Therefore, averages and standard deviations are not useful to summarize the data presented in section 8.2.2. Instead, regression is used to approximate each set of data by a power function, so that coefficients can be extracted. The results of the regression are shown in table 16.

	scaling factor	power	R^2
original system	328.496	-1.682	0.8692
EDA-system	866.401	-2.393	0.9418

Table 16: Cyclomatic Complexity regression results

The R^2 values show a reasonable fit for the EDA-system. The fit for the original system is less good, which is probably caused by a few outlying high values. Of the two coefficients, the power is the one to look at, since it is a measure for how quickly the frequencies decrease for higher Cyclomatic Complexity values. From a comparison between the two powers, the one for the EDA-system is a little smaller, indicating a faster decrease and therefore lower overall complexity.

The results from the Depth of Inheritance are also expected to follow a power law, because a larger depth is increasingly unlikely. Only a fraction of the ground-level classes will be inherited and only a fraction of the inheritors will be in turn inherited again and so forth. Therefore, just like for the Cyclomatic Complexity, regression is used to approximate each set of data by a power function. The results of the regression are shown in table 17.

	scaling factor	power	R^2
original system	54.566	-2.161	0.7645
EDA-system	104.230	-3.580	0.9998

Table 17: Depth of Inheritance regression results

The R^2 values show a good fit for the EDA-system. The fit for the original system not that good. This is probably caused by the use of libraries. Classes within the system can inherit from classes within a library, but the classes from the libraries are not included in the measurement results. This distorts the distribution somewhat. A comparison between the powers coefficients shows that the one for the EDA-system is quite a bit smaller. The before mentioned distortion of the original system's distribution could have made its coefficient a little larger, but the difference is most likely larger than this effect can account for. Therefore, the EDA-system also is somewhat simpler in this regard compared to the original system.

The two metrics give lower-level information, so some high-level observations regarding Simplicity are also included. In the EDA-system, the highest complexity is found within the EventStore module. Here, changes in state are processed and broadcast to other modules. Because events can arrive in a different order than the order in which they were sent, a large amount of possible but improbable situations can be distinguished. In this specific system, this was increased even more by two distinct types of possible state updates. Most updates rely on a previous state and therefore have an expected previous version of the state that is sent together with the update (see section 7.5.2). However, the updates that come from the SAPImport module work differently. These updates arrive from the data in another system and no knowledge about a previous version of the state in the EDA-system is included. Both types of updates need to be handled differently and their combination further increases the amount of possible situations. Many of these situations create circumstances that require extra effort to handle in the most desirable way. Since the majority of these situations are extremely unlikely, it would probably be best to simply reject the update for any situation that would otherwise require complicated actions. This is however an analysis based on experience. In the created EDA-system, the starting point was to be as general as possible with respect to the allowed situations. This meant that the system becomes more complex than it

would have needed to be.

In the original system, most complexity seems to occur within the MVC part of the system; in particular in the Controllers. The functionality is divided over the controllers partly based on the entity it concerns and partly based on the structure of the web application. This may increase the complexity of the design.

9.2.5. Self-descriptiveness

Since Self-descriptiveness is inherently a qualitative and somewhat subjective aspect, it is difficult to measure quantitatively. A qualitative difference in high-level Self-descriptiveness between the systems, is that the events used in the EDA-system carry some descriptive information within them. This means some additional information about what has previously happened in the system is included in the EDA-system compared to the original system. This translates into method names like handleCustomer-Created, which carry more meaning compared to, for example, a method called Index on a Controller in the original system's MVC part.

Another advantage of the EDA-system compared to the original system on the aspect of Self-descriptiveness and Understandibility in general, is the higher correspondence between the runtime and developmental division of the system. These divisions are shown respectively in the *process* and *development* views, used in sections 6 and 7. For the EDA-system, the *process* and *development* representations are very similar, which makes the way they are connected more intuitive than in the original system. As a result, understanding the runtime properties of the system based on developmental information is more straight-forward.

On the other hand, the EDA-system does not directly show what happens as a result of sending a message. Since recipients are not specified by the sender, it may be required to look inside all modules to find out where a particular event is received. In this regard, more effort is required to find out the complete set of actions taken by the entire system as result of some input.

9.2.6. Analysability and Diagnosability

A rough indication of both Analysability and Diagnosability is given by the Code Metric Maintainability Index. The results in figure 19 show that the EDA-system scored somewhat higher, but the scores lie relatively close together. As mentioned in section 8.2.5 the Maintainability Index has received criticism and should not be relied on too heavily. Furthermore, only a rough measurement of both systems is taken. Therefore, some qual-

itative observations about these two quality aspects are also given. These observations are derived from the experience of implementing the event-driven system and properties of the existing implementation of the original system.

Observations about Diagnosability

First of all, in the EDA-system, some communication is done by event sending instead of direct method calls. This means the origin of a system failure can be more difficult to determine because of the lack of a complete stack-trace. The stack-trace is cut off at the point of message communication. This means if the failure's real origin spans between modules, it becomes harder to track down. Also, if a message is not received by the intended modules, there is no direct sign that something went wrong. This could lead to problems staying undetected.

On the other hand, a stack-trace only gives information about what happened just before the failure occurred. If some record of all sent and received events is kept by the framework, it gives information about the entire history of the current system-run. This information can be useful when diagnosing failures that are caused by a particular chain of events within the system. The used Messageframework does contain some functionality in this direction, but a full exploration of this area lies beyond the scope of this project. It was therefore also not included in the EDA-system implementation.

Overall, which system is more easily diagnosed, depends on the types of failures that occur. For problems within a module, there are no real differences in Diagnosability between the systems. For problems between modules, it depends on how much history is needed to understand what is causing the problem. In the EDA-system, there is also the added difficulty that it is not always clear what part of the system causes the problem. For example, if certain actions as result of receiving a certain event in some module do not occur, the problem could lie with either the sending or the receiving module, or with the event-framework. Keeping records of all events would also make problems like this easier to diagnose.

In the EDA-system, the additional functionality for keeping records of all sent and received events also required some extra effort. First of all, the used event-framework needs to provide this functionality, or it has to be added manually. This functionality, when used, increases the processing time for events, since an additional record about them needs to be kept. Depending on the system, this could have some disadvantages. In the case of this EDA-system, it would mean processes like the SAP import process would take longer to complete, but it would most likely not cause any serious problems.

Observations about Analysability

The better high-level modularity makes it easier to determine which parts of the system need to be modified for a desired change in the EDA-system. The division into modules in this system is such that they each have a well defined part of the functionality, so as long as the event-sending system in between does not need to be changed (this type of change should be very unlikely once the system is totally developed), it is often very clear which modules are affected by a change in requirements. Consequently, the EDA-system is more analysable in this regard.

9.3. Audit trail

For this quality attribute, no measurements were conducted, so a qualitative comparison is made between both systems.

The logging of runtime information about what happens in the system is traditionally done by including log-statements in the source code that define what should be logged. Depending on the logging method, the messages are separated into different warning levels and they are written to file in some predefined format.

In an event-driven system like the EDA-system, this type of logging can still be applied within each module separately. This was for example done to mark the end of the state replay by the EventStore at system start-up. The difference between both types of systems is that, if the EDA modules exist in separate processes, there would be a separate log file for each module, as sharing the same file would lead to synchronization complications which are not desirable in an event-driven system. In the EDA-system, dividing the modules into separate processes ran into some problems (see section 10.1), so the same file could still be used here. The separation of log messages from different modules into different files could be an advantage or disadvantage, depending on what kind of information the logs contain. When enough information is included in the messages, multiple logs could be automatically combined into a single log and vice versa, so this difference would only constitute a little bit of extra effort.

Additionally, depending on the used message framework, the events in a event-driven system could themselves be used for audit purposes. The event framework contains information about which events are sent from which module and where they are received. If this information is saved into a log, it already provides a lot of information about what has happened in the system, without the need to explicitly include log-statements. As already mentioned in the part about Diagnosability in section 9.2.6, this possibility was not fully explored in the EDA-system.

Alternatively, a kind of meta-messages could be produced by the framework and received

by a separate logging module that combines and processes them into the kind of information that is wanted about the system during runtime. An actual implementation of this also lies outside of the scope of this project. A drawback of this approach compared to the direct use of information from within the used framework, is that a separate module does not have any information about the sender of the events and cannot detect messages being received.

9.4. Maintainability

For Maintainability aspects, different measurements and observations are used where applicable.

9.4.1. Repairability

For Repairability, in most of the same observations apply as for Diagnosability, since repairing a fault in the system first requires the problem to be located. Subsequently, the problem also has to be fixed. It is difficult to make general statements about this aspect, because it relies heavily on the type of problem encountered. Some observations were made on the EDA-system implementation:
An observed advantage is that the problem is usually located within a single module and therefore easier to fix. This especially reduces effort, because other modules do not need to be checked when looking for all the locations where the fix requires changes in the code. On the other hand, if the problem lies within the event-sending conventions, for example in the versioning system or the way the information is included in an event, changes are often required in many different places.

9.4.2. Modifiability and Extensibility

These two aspects both have to do with changes in requirements. Therefore they were both measured through change-scenarios, covered in section 8.1. The results are measurements of the amount of effort each change-scenario required in both systems. The effort was measured on several levels of detail. The results of the first change-scenario show comparable effort on all levels, except for the number of deleted lines, which is significantly larger for the EDA-system. The number of deleted lines is the least significant number, because deleting lines requires the least effort. This is especially true when the lines are consecutive, because the only effort lies in locating spots where lines need to be removed. The amount of removed lines mainly gives an idea about the effort needed for the reverse change. This means this change-scenario required comparable amounts of

effort in both systems. The reverse change would have required some more effort in the EDA-system than in the original system. This is mainly because the information sources needed for the pre-change situation are somewhat more separated in the EDA-system, because they are included in separate types of events. Receiving different types of events requires some additional code in the EDA-system, so changes where the borders between events are crossed in some way require more effort in the EDA-system. Other than that, the change in the first change-scenario was relatively easy in both systems.

The second change-scenario included a larger, more fundamental change. It shows that between the implementations of reduced size of both systems, the change in the original system already required a significantly larger amount of effort. The required changes were larger and somewhat more spread out. For an more extended implementation, the difference in required effort is expected to be even larger and more spread out. This change was therefore considerably more difficult in the original system than it was in the EDA-system.

The first change scenario can be entirely put into the Modifiability category, because it concerned changing the content of an already existing report. The second change scenario can be seen as a combination between Modifiability and Extensibility. The change in the internal workings of the system that allows for linked Customers falls into the Modifiability category, but the additional functionality to allow a user to link a Customer to another when entering details for it, can be seen as an extension to the system and therefore attributed to Extensibility. For more accurate measurements of both aspects, it would have been desirable to separate this change-scenario into two. This was however not feasible, because the data about the change in the original system were too much entangled.

Overall, the measurements show that the EDA-system scores somewhat higher on both Modifiability and Extensibility, but it does depend on the type of change or extension.

9.4.3. Scalability

To determine Scalability characteristics, the behaviour of the two systems under increased size is analysed. Ways in which an increase in size is relatively likely, is first of all the amount of data that is stored and processed. This would for example mean more Areas, SalesPersons and/or Customers. All of these would mean extra storage space required in both systems.

Specifically for an increase in customers (per area), the time and resource behaviour dependencies can more quantitatively be determined, because the time and resource measurements were done for different areas.

For the original system, the dependence of the load time on the number of customers (see table 7) is shown in figure 30. A clear correlation can be seen where for each extra customer, about 0.006 extra seconds of load time are added.

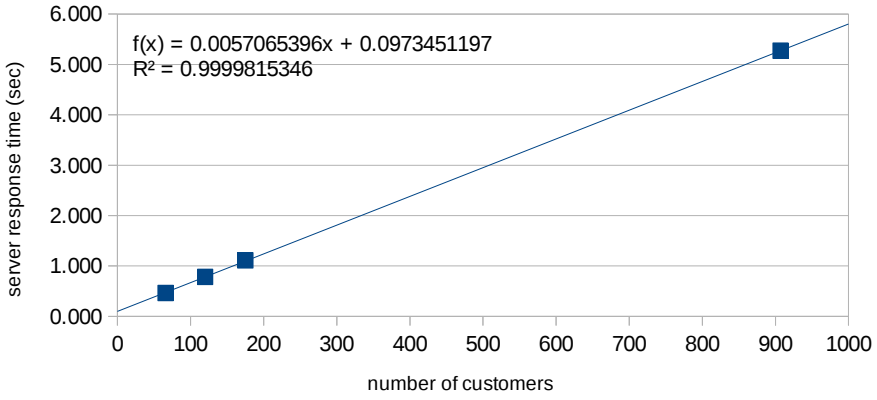

Figure 30: Original system – server response time versus number of customers

For the EDA-system, figures 22 and 23 and table 11 all indicate no correlation between the server response times and the number of customers. To show this even more clearly, figure 31 contains the corresponding graph. The R^2 value shows the absence of a correlation.

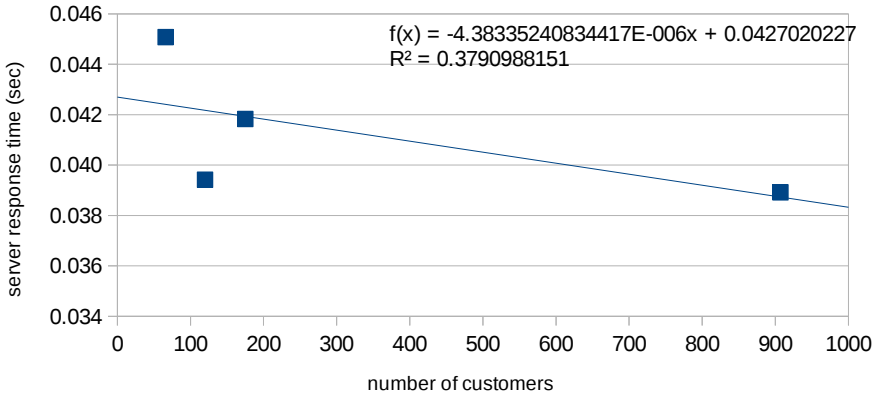

Figure 31: EDA-system – server response time versus number of customers

Regarding load times, the EDA-system is therefore much more scalable. An increasingly large number of customers per area in the original system would mean increasingly large load times, while this would not be the case in the EDA-system. On the other hand, for the start-up time of the system, this dependence is the other way around. In the EDA-system, it is dependent on the amount of data in the system, while for the original

system it not. This is however of less consequence, because a restart of the system is only required under special circumstances.

The dependence on number of customers of the memory usage for the original system lies only in the peak memory usage. This dependence is shown in figure 32. About 4826 extra bytes of temporary memory are needed per customer. The increase in ground level memory usage is independent of the number of customers.

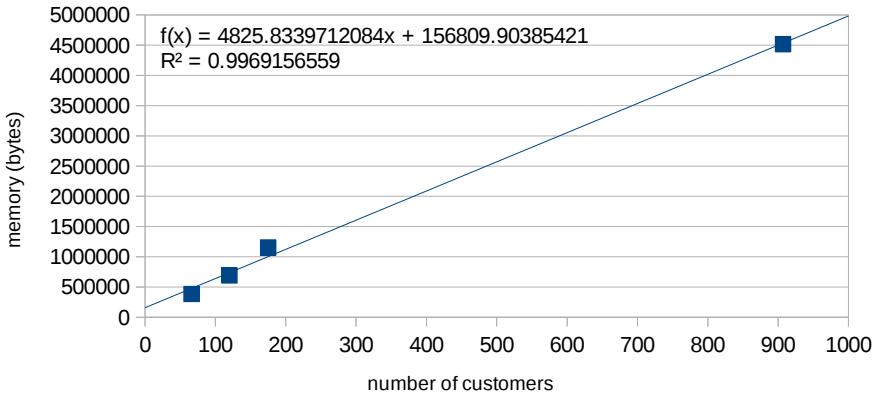

Figure 32: Original system – memory usage peak heights versus number of customers

As noted is section 9.1.2, for the EDA-system, the ground level memory usage *is* dependent on the number of customers. This dependence is shown in figure 33. For each additional customer, the increase in memory usage is about an extra 300 bytes. As also mentioned in section 9.1.2, this extra memory is eventually released, so the memory effects are not long term.

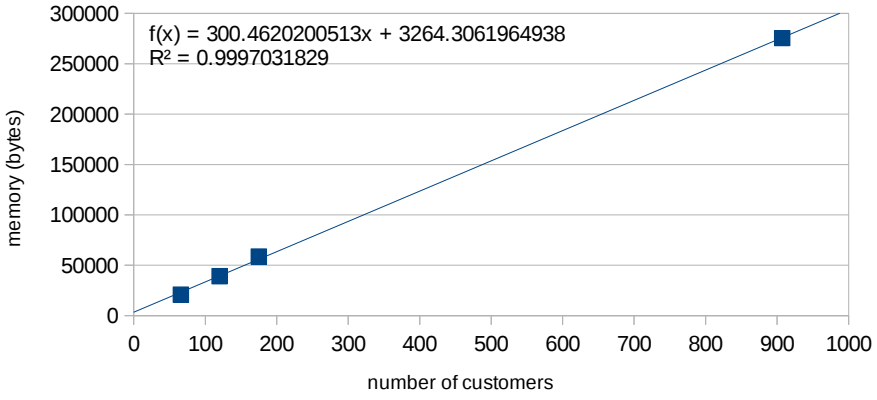

Figure 33: EDA-system – memory usage increase versus number of customers

Regarding memory usage, the size dependent effect are only short term. For the original systems, the dependence on the number of customers is larger, but also of much shorter duration. The overall relative scalability of both systems for this aspect is therefore dependent on how the system is used. Additionally, both systems scale relatively well in this regard, since resource problems are only expected to occur for a very large increase in size.

For all discussed scalability aspects, a not yet mentioned difference is that in the EDA-system, time and resource behaviour can be improved by adding additional nodes. For the original system, this is significantly more difficult. Regarding disk storage, SQL databases like the one used in the original system, cannot easily be distributed between different nodes [36]. For the EventStore used in the EDA-system, this is easier, because there are far fewer constraints on it. Regarding time and memory behaviour, in the EDA-system each module can be run on a different node, with events travelling between them. In the original system, distribution over different nodes require a significant amount of extra effort, because the communication happens by direct method calls. Additionally, a good division of processing effort over the different nodes is not immediately apparent. Overall, the EDA-system therefore has better horizontal scalability.

9.4.4. Testability

Testability was also not measured in a quantitative way. However, tests were made for both systems. Based on these, a qualitative comparison is made.

The higher level tests in the EDA-system take the form of separate testing modules. Simple tests can be performed by sending a few events into the system and receiving some events back to see whether the expected behaviour happens. This method also makes it easy to test separate modules, because the tests can simply be run by only including the modules to be tested, together with the test module(s). On the other hand, tests that require a more complicated sequence of actions to happen, can become more complex, making it harder to verify if the test itself is still correct. This can happen especially in cases where the events sent by the test module are dependent on received events, which was the case in many tests in the EDA-system, because of version-related reasons. Additionally, these tests cannot be executed automatically by existing testing tools, so they require a little more effort to run. Also, the tests require a little more effort initially, because the testing module outline first needs to be constructed before actual tests can be made.

10. Conclusions

In this section, the comparisons from the case-study are generalized to arrive at broader statements about the use of EDA, which are part of the research goal. First of all, the used event-framework provided the case-study EDA-system with a specific set of possibilities. In order to make statements about EDA in general, it should be considered that other event-frameworks might have different options or limitations. These are explained in section 10.1.

Additionally, the two systems differ on multiple levels. First of all, the EDA-system uses messages for top-level communication, while the original system uses only method calls throughout the system. Secondly, the original system is totally synchronous, while the EDA-system is mostly asynchronous in the communication between modules. Finally, some characteristics are specifically part of the EDA pattern. For a more accurate analysis, the results of the comparison are attributed to the level/aspect that most likely causes them (sections 10.2, 10.3, 10.4).

The other part of the research goal is to collect recommendations about the use of EDA. These are presented in section 10.5. Finally, section 10.6 highlights possibilities for future research on this topic.

10.1. Generalization from used Messageframework

First of all, the used Messageframework has some limitations where other available frameworks provide a broader set of possibilities, influencing the maintainability results from the case-study.

The used Messageframework provided functionality for creating and receiving messages. In creating messages, data could be included under different headers. Messages are received per message-type and additional filters can be specified on properties of the event contents. A limitation regarding this, is that the filters must be specified as literal constants, so they cannot be determined with the use of any variables. This meant that it was, for example, not possible to make a module template and parametrize it with, for instance, an area number in order to only receive events regarding that area. Other event-frameworks also have filters based on type and/or content of events ([37, 38], which are expected to be less static. Also, some frameworks (e.g. [37]) treat events more in terms of streams, with operations that can be applied to the stream as a whole, providing more options in how to use events.

The Messageframework provides the possibility to separate the modules into different nodes (separate processes instead of separate execution threads within the same process),

by making a separate Visual Studio solution for each module. This also provides the option to distribute the modules over several different machines. However, when applying this to the implementation of the EDA-system made during the case-study, problems were encountered. Some of the problems were the maximum amount of data allowed within a single event and incorrect processing of events containing certain characters. These problems could not be fixed without changing the Messageframework itself, which lies outside of the scope of this project. Therefore, a node separation was not achieved in the built implementation and runtime behaviour when using this aspect of EDA could not be observed. However, high confidence assumptions about how it would have worked could still be made, so this possibility is already included in the comparisons in section 9. Other event-frameworks, like [38, 39], are available that provide this option, most likely without the same problems.

Finally, the Messageframework did not provide any options for Complex Event Processing (CEP). CEP is the aggregation and combination of multiple events that are related in some way in content and/or timing, to create new higher-level events [40]. This functionality was not needed in the case-study system, but there are event-frameworks available that do provide this functionality [40].

Overall, the used Messageframework suffices for the implementation of the case-study EDA-system, but other frameworks exist that provide a broader set of options on how to deal with events in an EDA. More supported possibilities usually lead to better maintainable systems, because any functionality already provided does not need to be implemented in the system itself. It also separated functionality concerning the communication principles from the business functionality of the implemented system. This means the use of EDA could lead to more maintainable systems than would be expected based on the case-study results.

Additionally, the used framework has an impact on the timing properties of systems using it. The used framework had some limitations regarding throughput; when the generated events were sent too close to each other in time, not all of them would arrive at the other side. This had a negative effect on for example import times, as a small delay had to be inserted to make sure all events arrived everywhere in the system. Other frameworks are expected to have better properties regarding throughput, but this could not be verified from literature research alone.

10.2. Differences attributed to use of messages

In this context, we define a message to be a bit of information that is sent from one part of a system and received in another part, where it triggers a response. There is no inherent coordination, continuation or context preservation between the two parts

[1]. The message could be directed towards a specific location, or could be sent to any location that has declared to be interested in it. It can specifically request some action to be taken by the receiving end or be purely descriptive.

The first comparison observation that is caused by the use of messages for top-level communication in the EDA-system, instead of the method calls used in the original system, is the lack of a complete stack-trace. This difference is discussed in the part about Diagnosability in section 9.2.6. It means that part of the observed differences in the ease of determining the source of a problem are due to this difference in communication method. A non-EDA method that used messages instead of method calls would also not have a complete stack-trace to track a failure; it would be cut off wherever a message is used instead of a method-call. On the upside, such a non-EDA method would also have the possibility of saving all messages sent in the system, providing more information to use when diagnosing. However, if the messages were not as descriptive as events generally are, it would still give less information than in an EDA system.

The lack of Class Coupling between classes in different modules can also be attributed to the use of messages (as long as they are not embedded into the *C#* language). However, just messages instead of real events can still cause a relatively high amount of coupling, because messages can be directive, directed towards a specific target and expect a reply. Systems using messages are therefore not necessarily more modular than systems using calls.

The construction of an audit-log by collecting all messages sent in the system would also be possible in a non-EDA system using messages. This aspect of improved Audit trail (see section 9.3) capabilities of an EDA system could therefore partly be contributed to the message aspect. However, general messages are not necessarily as descriptive as events, and give a less useful view of the runtime history of a system. Also, in order to use the method of constructing a separate logging module is only possible if messages are broadcast throughout the system instead of directed towards an intended receiver, which is not a required property of a system using message communication.

10.3. Differences attributed to asynchronous communication

The communication used in the case-study EDA-system is asynchronous in nature. Once an event is sent, the sending module continues on without waiting for the message to be received by any receiving modules. A property of the implementation that can be attributed to this asynchronous communication, is the unknown order of arrival of events. When a sending modules sends multiple messages in quick succession without waiting

for a sign of arrival, they can arrive in a different order. This has some consequences on Simplicity, as discussed in section 9.2.4. This type of extra complexity may also be experienced in non-EDA systems with asynchronous communication.

Also, some part of the communication between the front-end and the storage module in the EDA-system needs to be synchronous, because it is desirable to know whether an update done by the user is accepted and processed before moving on and doing possibly another update on the same data. The modules are manually synchronized in these situations by explicitly making the front-end wait for a specific event before moving on. This explicit implementation makes the system somewhat less maintainable, first of all simply because there is a need for extra code, but also because there may be some unanticipated corner cases that are not handled correctly by this approach in the current or a future version of the system. This means that asynchronous systems in general decrease in maintainability somewhat if there is still a need for some form of synchronous communication.

Additionally, asynchronous communication also leads to some uncertainty, because a sender does not know whether its message has been received (yet). In other words, the reduced error detection can be attributed to it. Message arrival could be tracked by the used event-framework, or handled by some other method, but at a minimum, it means some extra effort needs to be put in if certainty is required about the arrival of messages.

On the other hand, asynchronous communication has a (usually quite large) positive effect on parallelization and therefore on timing behaviour and (horizontal) scalability. Whenever two parallel executions need to synchronize, they need to wait for each other, leading to less actual parallel computation. This problem is not encountered when using asynchronous communication. Therefore, the better horizontal scalability of the EDA-system can for a large portion be contributed to the used asynchronous communication. This is also the case for the responsiveness of the system, because different parts do not need to wait for each other.

Furthermore, asynchronous communication makes the different parts of the system less connected and more stand-alone. The improved Testability, as discussed in section 9.4.4, can be partly attributed to this.

10.4. Differences attributed to event-characteristics

Most of the remaining differences can be attributed to the specific characteristics of EDA. First of all, EDA dictates that modules should have no knowledge about each other whatsoever. This contributes to maintainability, as discussed in section 9.4.

Secondly, EDA is push-based; changes drive actions and are immediately propagated

throughout the system. Most of both the time and resource behaviour differences, discussed in section 9.1 can be attributed to this. The system completes actions when a change occurs, instead of when something is requested. This means that request completion times are low in an event-driven system, since all needed information should already be present in the module in which the request occurs. At the same time, it means this module keeps a copy of all data that is needed within it. Considering the timing aspect, the difference may not be substantial for systems that still require relatively large calculations to be performed on the data within a module. This is because EDA does not specify anything about the time at which intra-modular calculations should be executed. If they are done on demand, it would still result in relatively long waiting times.

However, for a system where most of the completion time of actions is spent gathering the necessary information together, EDA will perform well in comparison to the traditional alternative. This is also the case in the case-study EDA-system, where reports are often just a selection of data arranged in an practical way for users. For some reports, adding monetary values together is required, but this is a relatively small portion of all the needed operations in building the report. Therefore, the measured load times are substantially smaller in the EDA-system, while the memory usage is considerably larger. In many systems, trading in extra memory for shorter load times is beneficial, since the cost of extra memory is relatively low nowadays. Especially if it can be added in the form of extra machines (horizontal), which is usually the case with EDA. To be precise, the total amount of processing time is not reduced, but shifted to different moments. When a change occurs, an EDA system takes slightly longer to process it, but this time is saved when a (user-)request is made.

The observed extra descriptiveness of the EDA-system can be completely attributed to EDA, since the events contain descriptive information about developments during runtime. In addition, the specific characteristics of EDA cause the system to have clear connections between the developmental structure and the runtime structure, as discussed in section 9.2.5. At the same time, the fact that events in EDA are not directed towards a recipient, but broadcast throughout the system, makes it more difficult to find out the cause and effect relationships in an EDA system. This causes the system to be more difficult to understand and analyse.

10.5. Recommendations

The first recommendation in the consideration about whether to use EDA is the following. The push-based approach of EDA is only beneficial if the needed information for the possible actions/requests is known beforehand in the system. This is not the case if, for example, users enter their own custom queries to be executed on the data. Even

if the framework allows for the message subscriptions to be changed during runtime, events that were previously not of interest, but are now, have not been received, so a user query cannot be executed in this way. In the case of such a system, no good module separation can be found for EDA, because it is not possible to successfully separate the front-end from the data storage. The EDA pattern is not a good fit for this type of system.

Another point may be the importance of time and resources. For systems that are tight on memory and/or storage space, EDA is not a good fit. The same goes for systems where data can be requested in a large variety of ways. The mentioned custom queries are an extreme case of this, but even when the data for each request *can* be prepared in advance, it may not be worthwhile when there are a large number of possible requests that are in practice requested very infrequently. In that case, it is most likely better to use a pull-based set-up.

When EDA has already be decided on, there are also some things to keep in mind to make implementing this pattern more successful. First of all, it is beneficial to have the central data storage of the system keep as much information as possible. This way, any future requirement changes in data requests and processing do not require changes in the storage part of the system, making such changes less costly. An example of this effect is seen in change-scenario 2 (section 8.1.2). These types of changes are expected to be more likely than changes in the way data is supplied in the system. The latter type of change would still be relatively expensive to make.

In addition, the whole set-up around event sending and arrival is best kept as general as possible. Changing these aspects requires changes in many parts of the system, making them expensive. For example, parts like a versioning system, which is required in one form or another in many EDA systems, should be well thought out. It is beneficial if it can support a variety of possible situations, possibly by including more information, even if it seems unnecessary at first.

Using an event store as data storage mechanism fits in well with the properties of EDA and should be considered. When using it, it is advantageous to consider the types of input events that should be stored and imagine all general types of combinations between them. If handling some rare combinations requires complicated solutions, it may be better to reject these situations. This can be done by labelling certain events as rejected when they occur in such a combination. For example in the built EDA-system, it would have been beneficial to reject no-ops, i.e. updates where nothing changes, instead of allowing them. Allowing them turned out to cause multiple complications and made the system more complex.

10.6. Future work

Further research on the topic of EDA could include doing comparisons for more cases, so better generalizations can be made. This would ideally include using different event frameworks to see if there are significant differences and to give indications about which ones are better suited for which kinds of problems. Also, other quality attributes could be included in the comparison. For example, Security was excluded in this study, but could potentially be interesting for other systems where EDA is considered in the development stage.

It would also be beneficial to look at the benefits of Complex Event Processing, since it likely adds very powerful possibilities for an event-driven system.

Bibliography

[1] G. Hohpe. "Programming without a call stack – event-driven architectures". In: *Objekt Spektrum* (2006).

[2] B. M. Michelson. "Event-driven architecture overview". In: *Patricia Seybold Group* 2 (2006).

[3] URL: `http://www.sogyo.nl/`.

[4] L. Filipponi et al. "Smart City: An Event Driven Architecture for Monitoring Public Spaces with Heterogeneous Sensors". In: *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*. 2010, pp. 281–286.

[5] O. Etzion. "Towards an Event-Driven Architecture: An Infrastructure for Event Processing Position Paper". In: *Rules and Rule Markup Languages for the Semantic Web*. Ed. by A. Adi, S. Stoutenburg, and S. Tabet. Vol. 3791. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 1–7.

[6] S. Y. Ghalsasi. "Critical Success Factors for Event Driven Service Oriented Architecture". In: *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ICIS '09. ACM, 2009, pp. 1441–1446.

[7] D. Garlan and M. Shaw. "An introduction to software architecture". In: *Advances in Software Engineering and Knowledge Engineering*. World Scientific Publishing Company, 1993, pp. 1–39.

[8] M. Shaw and P. Clements. "The golden age of software architecture". In: *Software, IEEE* 23.2 (Mar. 2006), pp. 31–39.

[9] D. Garlan, R. Allen, and J. Ockerbloom. "Architectural mismatch or why it's hard to build systems out of existing parts". In: *Software Engineering, 1995. ICSE 1995. 17th International Conference on*. IEEE. 1995, pp. 179–179.

[10] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Pearson Education India, 2007.

[11] P. Bengtsson. "Towards maintainability metrics on software architecture: An adaptation of object-oriented metrics". In: *First Nordic Workshop on Software Architecture (NOSA'98), Ronneby*. Vol. 10. 1998.

[12] P. B. Kruchten. "The 4+1 view model of architecture". In: *Software, IEEE* 12.6 (1995), pp. 42–50.

[13] ISO. *Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models*. ISO/IEC 25010:2011. Geneva, Switzerland: International Organization for Standardization, 2011.

[14] J. A. McCall, P. K. Richards, and G. F. Walters. *Factors in Software Quality. Volume-III. Preliminary Handbook on Software Quality for an Acquisiton Manager*. Tech. rep. DTIC, 1977.

[15] M. Barbacci et al. *Quality Attributes*. Tech. rep. DTIC Document, 1995.

[16] B. W. Boehm, J. R. Brown, and M. Lipow. "Quantitative evaluation of software quality". In: *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press. 1976, pp. 592–605.

[17] R. B. Grady and D. L. Caswell. *Software metrics: establishing a Company-wide program*. Prentice Hall, 1987.

[18] R. G. Dromey. "Cornering the chimera". In: *IEEE Software* 13.1 (1996), pp. 33–43.

[19] M. Ortega, M. Pérez, and T. Rojas. "Construction of a systemic quality model for evaluating a software product". In: *Software Quality Journal* 11.3 (2003), pp. 219–242.

[20] E. Mnkandla and B. Dwolatzky. "Defining Agile Software Quality Assurance". In: *International Conference on Software Engineering Advances*. 2006, pp. 36–36.

[21] H. Breivold, I. Crnkovic, and P. Eriksson. "Analyzing Software Evolvability". In: *32nd Annual IEEE International Computers, Software and Applications Conference (COMPSAC)*. 2008, pp. 327–330.

[22] F. Losavio et al. "Quality characteristics for software architecture". In: *Journal of Object Technology* 2.2 (2003), pp. 133–150.

[23] M. Broy and E.-R. Olderog. "Trace-oriented models of concurrency". In: *Handbook of process algebra* (2001), pp. 101–195.

[24] M. G. Hinchey. "A formal design method for real-time Ada software". In: *Ada: Towards Maturity* (1993), pp. 123–137.

[25] URL: `https://www.httpwatch.com/`.

[26] X. Liu and Q. Wang. "Study on application of a quantitative evaluation approach for software architecture adaptability". In: *Quality Software, 2005. (QSIC 2005). Fifth International Conference on*. Sept. 2005, pp. 265–272.

[27] R. Kazman et al. “SAAM: A method for analyzing the properties of software architectures”. In: *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press. 1994, pp. 81–90.

[28] P. Bengtsson and J. Bosch. “Scenario-based software architecture reengineering”. In: *Software Reuse, 1998. Proceedings. Fifth International Conference on*. IEEE. 1998, pp. 308–317.

[29] URL: `http://nhibernate.info/`.

[30] F. F.-H. Nah. “A study on tolerable waiting time: how long are Web users willing to wait?” In: *Behaviour & Information Technology* 23.3 (2004), pp. 153–163.

[31] URL: `http://signalr.net/`.

[32] URL: `http://owin.org/`.

[33] URL: `https://geteventstore.com/`.

[34] P. Oman and J. Hagemeister. “Construction and testing of polynomials predicting software maintainability”. In: *Journal of Systems and Software* 24.3 (1994), pp. 251–266.

[35] I. Heitlager, T. Kuipers, and J. Visser. “A Practical Model for Measuring Maintainability”. In: *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*. IEEE. Sept. 2007, pp. 30–39.

[36] R. Cattell. “Scalable SQL and NoSQL Data Stores”. In: *ACM SIGMOD Record* 39.4 (May 2011), pp. 12–27.

[37] E. Meijer. “Reactive extensions (Rx): curing your asynchronous programming blues”. In: *ACM SIGPLAN Commercial Users of Functional Programming*. ACM. 2010, p. 11.

[38] P. R. Pietzuch and J. M. Bacon. “Hermes: A distributed event-based middleware architecture”. In: *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. IEEE. 2002, pp. 611–618.

[39] J. Armstrong. “Erlang”. In: *Commun. ACM* 53.9 (Sept. 2010), pp. 68–75.

[40] D Robins. “Complex event processing”. In: *Second International Workshop on Education Technology and Computer Science. Wuhan*. 2010.

[41] DoD. *Department of Defense Trusted Computer System Evaluation Criteria*. DoD Standard 5200.28-STD. U.S. Department of Defence, 1985.

Appendices

A. Definitions of quality criteria

Table 18 gives the definitions used in this research project for all considered quality criteria. The different sources these definitions were adapted from, are referenced by number in the table. The numbers stand for the following:

[1] adapted from [10]
[2] adapted from [13]
[3] adapted from [14]
[4] adapted from [20]
[5] adapted from [21]
[6] adapted from [16]
[7] adapted from [41]

Functionality	the ability of the system to do the work for which it was intended[1]
Completeness	degree to which the set of provided functions covers all the specified tasks and user objectives[2]
Appropriateness	degree to which the functions facilitate the accomplishment of specified tasks and objectives[2]
Traceability	degree to which the system provides a thread from the requirements to the implementation with respect to the specific development and operational environment[3]
Correctness / Conformance	degree to which the system satisfies its defined specifications[3]
Suitability	degree to which the system provides functions that meet stated and implied needs when used under specified conditions[2]
Usability	degree to which a system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use[2]
Learnability	degree to which a system can be used by specified users to achieve specified goals of learning to use the system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use[2]

User Error Protection	degree to which a system protects users against making errors[2]
Attractiveness	degree to which a user interface enables pleasing and satisfying interaction for the user[2]
Communicativeness	degree to which the system provides useful inputs and outputs which can be assimilated[3]
Appropriateness Recognizability	degree to which users can recognize whether a system is appropriate for their needs[2]
Operability	degree to which a system has attributes that make it easy to operate and control[2]
Portability	degree of effectiveness and efficiency with which a system or component can be transferred from one hardware, software or other operational or usage environment to another[2]
Replaceability	degree to which a system can replace another specified software system for the same purpose in the same environment[2]
Installability	degree of effectiveness and efficiency with which a system can be successfully installed and/or uninstalled in a specified environment[2]
Software System Independence	degree to which the software system is dependent on the software environment (operating systems, utilities, input/output routines, etc.)[3]
Machine Independence	degree to which the software system is dependent on the hardware system[3]
Adaptability	degree to which a system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments[2]
Security	degree to which a system protects information and data so that persons or other systems have the degree of data access appropriate to their types and levels of authorization[2]
Integrity	degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data[2]
Accountability	degree to which the actions of an entity can be traced uniquely to the entity[2]

Confidentiality / Privacy	degree to which a system ensures that data are accessible only to those authorized to have access[2]
Authenticity	degree to which the identity of a subject or resource can be proved to be the one claimed[2]
Non-repudiation	degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later[2]
Reliability	degree to which the system or component performs specified functions under specified conditions for a specified period of time[2]
Fault Tolerance	degree to which the system or component operates as intended despite the presence of hardware or software faults[2]
Recoverability	degree to which, in the event of an interruption or a failure, the system can recover the data directly affected and re-establish the desired state[2]
Maturity	degree to which a system or component meets needs for reliability under normal operation[2]
Accuracy / Precision	degree to which the software or component provides the required precision in calculations and outputs[3]
Robustness	appropriate performance of the system under cases not covered by the specification[4]
Availability	degree to which a system or component is operational and accessible when required for use[2]
Maintainability	degree of effectiveness and efficiency with which a system can be modified by the intended maintainers; modifications can include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications[2]
Repairability	effort required to locate and repair an error in an operational program[3]
Modifiability	degree to which a system can be effectively and efficiently modified without introducing defects or degrading existing quality[2]

Evolvability	the ability of a system to accommodate changes in its requirements throughout the systems lifespan with the least possible cost while maintaining architectural integrity[5]
Extensibility / Augmentability	degree to which the software system enables the implementation of extensions to expand or enhance the system with new capabilities and features with minimal impact to the existing system[5]
Testability	degree of effectiveness and efficiency with which test criteria can be established for a system or component and tests can be performed to determine whether those criteria have been met[2]
Compatibility	degree to which a system or component can exchange information with other systems or components, and/or perform its required functions, while sharing the same hardware or software environment[2]
Interoperability	degree to which two or more systems or components can exchange information and use the information that has been exchanged[2]
Co-existence	degree to which a system can perform its required functions efficiently while sharing a common environment and resources with other systems, without detrimental impact on any other system[2]
Reusability	degree to which an asset can be used in more than one system, or in building other assets[2]
Generality	degree to which the software provides breadth in the functions performed[3]
Self-containedness	degree to which a system or component performs all its explicit and implicit functions within itself[6]
Audit trail	degree to which actions performed during the system runtime can be reproduced[7]
Efficiency	performance relative to the amount of resources used under stated conditions[2]
Performance / Timeliness	degree to which the response and processing times and throughput rates of a system, when performing its functions, meet requirements[2]

Capacity	degree to which the maximum limits of a system parameter meet requirements[2]
Resource Behaviour	degree to which the amounts and types of resources used by a system, when performing its functions, meet requirements[2]
Structuredness	degree to which the system possesses a definite pattern of organization[6]
Modularity	degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components[2]
Consistency / Uniformity	degree to which uniform design and implementation techniques and notation are used[3]
Conciseness	degree to which the implementation of functionality is provided with a minimum amount of code[3]
Understandability	degree to which the purpose of different parts of the system is clear to the inspector[6]
Analyzability	degree of effectiveness and efficiency with which it is possible to assess the impact on a system of an intended change to one or more of its parts, or to diagnose a system for deficiencies or causes of failures, or to identify parts to be modified[2]
Simplicity	degree to which the software system provides functionality in the most understandable manner[3]
Self-descriptiveness	degree to which the code contains enough information for a reader to determine or verify its objectives, assumptions, constraints, inputs, outputs, components, and revision status[6]
Legibility	degree to which the code's function is easily discerned by reading the code[6]

Table 18: Definitions of quality criteria

B. Full measurement data

Change-scenario data

Tables 19, 22, 21, 20 and 23 show change-scenario results. They express the effort required for each change-scenario in terms of modules, namespaces, classes and added and deleted lines. The bottom row of each table gives the total values for each column.

modules	**namespaces**	**classes**	**lines**	
			+	−
SalesPortal	Models	CIFIndexModel	11	19
	Controllers	CustomerController	1	5
	Views	Customer/Cif	10	6
Domain.Sales	Services	CoopModelService	0	15
2	4	4	22	30

Table 19: Original system (reduced) – change scenario 1

modules	**namespaces**	**classes**	**lines**	
			+	−
Domain.Sales	Entities	Customer	263	171
		BudgetProposal	18	0
		Turnover	20	1
		AreaCustomer	258	-
		Contact	3	0
	Services	BurgetProposalService	40	0
	Repositories	IAreaCustomerRepository	12	-
		IBudgetProposalRepository	7	0
		ITurnoverRepository	1	0
Repository .NHibernate	Repository .NHibernate	AreaCustomerRepository	21	-
		CustomerRepository	18	16
		BudgetProposalRepository	12	2
		TurnoverRepository	9	0
SalesPortal	Controllers	CustomerController	326	93
	Views	Customer/Edit	27	27
3	6	15	1035	310

Table 20: Original system (reduced) – change scenario 2

modules	**namespaces**	**classes**	**lines**	
			+	−
Domain.Sales	Entities	Customer	263	171
		BudgetProposal	18	0
		Turnover	20	1
		AreaCustomer	258	-
		Contact	3	0
	Services	BurgetProposalService	40	0
	Repositories	IAreaCustomerRepository	12	-
		IBudgetProposalRepository	7	0
		ITurnoverRepository	1	0
		INpsRepository	1	1
Repository	Repository	AreaCustomerRepository	21	-
.NHibernate	.NHibernate	CustomerRepository	18	16
		BudgetProposalRepository	12	2
		TurnoverRepository	9	0
		NpsRepository	42	36
SalesPortal	Controllers	CustomerController	326	93
		ReportController	4	0
		BudgetController	1	1
		ProposalOverviewController	1	0
	Views	Customer/Edit	27	27
batchpdfcreator	BatchPdfCreator	PdfGenerator	6	12
		Program	14	8
	TableClasses	TableFactory	10	3
4	8	23	1114	371

Table 21: Original system (*) – change scenario 2

modules	**namespaces**	**classes**	**lines**	
			+	−
CIFIndexList	CIFIndexList	MessageConvHelper	0	12
		CIFIndexList_Payload	4	72
		StateManager	0	68
		BPModelTransformer	-	24
		BudgetProposalModel	-	31
		CIFIndexModel	2	209
		CIFIndexTransformer	5	0
Gateway	FrontEnd scripts	ciftable.js	4	26
2	2	7	15	442

Table 22: EDA-system – change scenario 1

modules	**namespaces**	**classes**	**lines**	
			+	−
EventStore	EventStore	EventNames_Store	2	0
		Customer_Store	15	0
		EventStorage	34	13
		EventStoreWrapper	173	20
		EventStore_Payload	2	1
		DataExtracter	16	4
		MessageSender	10	0
		StateBuilder	20	21
		ImportStorage	2	2
Gateway	Gateway	Gateway_Payload	28	3
	FrontEnd_SignalR	CIFHub	7	0
	FrontEnd scripts	index.js	27	13
		cifform.js	1	0
Helper	Helper	EventNames_Message	5	4
		MessageConversion	14	6
3	5	16	356	87

Table 23: EDA-system – change scenario 2

Code Metrics data

The complete frequency results for some of the code metrics are shown in table 24. Values for which the frequencies in both systems were zero are omitted from the table.

Class Coupling	Original system	EDA-system	Cyclomatic Complexity	Original system	EDA-system
0	11	18	1	790	812
1	5	3	2	135	145
2	6	10	3	55	76
3	11	12	4	38	50
4	8	8	5	32	26
5	8	6	6	14	14
6	12	6	7	12	11
7	6	5	8	14	3
8	6	7	9	6	6
9	5	6	10	6	3
10	4	2	11	7	1
11	6	2	13	1	1
12	2	4	14	1	0
13	2	2	16	1	0
14	1	2	17	0	1
15	2	2	21	1	0
16	0	3	25	0	1
17	2	1	42	1	0
19	1	1	46	1	0
20	1	1	49	1	0
21	1	0	59	1	0
22	1	0			
23	2	0	Depth of	Original	EDA-
24	1	3	Inheritance	system	system
25	0	1	1	96	103
26	1	0	2	11	9
27	0	3	3	1	2
28	1	0	4	4	0
29	1	0	5	3	0
30	2	0	6	0	0
32	0	1	7	1	0
33	1	0			
34	0	0			

35	0	1
41	1	1
42	0	1
43	0	1
45	1	0
46	1	0
47	0	1
50	1	0
67	1	0
76	1	0

Table 24: Code metrics frequency results

Time measurement data

Tables 25 and 26 show the time measurements for the original system. For each measurement, the report for the respective Area is loaded and both the HTTP load time and the server response time in seconds are measured. Table 27 shows the time measurements for the EDA-system. Here, for each measurement, the report for the respective Area is also loaded, but here, the server response time and the HTTP load time were the same (see section 8), so only one measured time is shown per measurement point.

Run 1 (time)			**Run 2** (time)		
Area	HTTP load time (sec)	server response time (sec)	Area	HTTP load time (sec)	server response time (sec)
100	5.936	5.631	20	1.468	1.336
100	5.625	5.349	20	1.280	1.141
100	5.639	5.274	20	1.258	1.121
20	1.330	1.098	100	5.645	5.362
20	1.305	1.145	100	5.691	5.396
20	1.263	1.100	100	5.709	5.412
63	0.876	0.757	34	0.586	0.472
63	0.902	0.780	34	0.589	0.474
63	0.916	0.813	34	0.584	0.468
34	0.564	0.476	63	0.911	0.783
34	0.580	0.470	63	0.911	0.785
34	0.554	0.444	63	0.906	0.789

Table 25: Original system – time measurement data, runs 1 and 2

Run 3 (time)			**Run 4** (time)		
Area	HTTP load time (sec)	server response time (sec)	Area	HTTP load time (sec)	server response time (sec)
63	1.112	0.663	34	0.772	0.992
63	0.905	0.448	34	0.555	0.748
63	0.944	0.460	34	0.566	0.787
34	0.619	0.776	63	0.895	0.448
34	0.563	0.793	63	0.917	0.454
34	0.633	0.785	63	0.906	0.483
100	5.555	1.088	20	1.231	5.270
100	5.512	1.079	20	1.230	5.238
100	5.529	1.060	20	1.195	5.252
20	1.330	5.146	100	5.438	1.152
20	1.273	5.102	100	5.388	1.099
20	1.270	5.186	100	5.467	1.115

Table 26: Original system – time measurement data, runs 3 and 4

Run 1 (time)		**Run 2** (time)		**Run 3** (time)		**Run 4** (time)	
Area	time (sec)	Area	time (sec)	Area	time (sec)	Area	time (sec)
100	0.056	20	0.052	63	0.023	34	0.052
100	0.071	20	0.056	63	0.032	34	0.052
100	0.026	20	0.021	63	0.056	34	0.046
20	0.045	100	0.059	34	0.050	63	0.063
20	0.049	100	0.033	34	0.061	63	0.059
20	0.009	100	0.033	34	0.029	63	0.023
63	0.029	34	0.066	100	0.025	20	0.065
63	0.021	34	0.018	100	0.032	20	0.059
63	0.050	34	0.058	100	0.042	20	0.024
34	0.022	63	0.036	20	0.028	100	0.032
34	0.049	63	0.027	20	0.029	100	0.027
34	0.038	63	0.054	20	0.065	100	0.031

Table 27: EDA-system – sever response time measurement data

Memory measurement data

Table 28 shows the memory measurements for the original system. The measurement points represent the memory just after loading the report page for the respective Area. The measurement points with no Area are points where the welcome-page is loaded to get ground level memory usage points. Tables 29 and 30 show the memory measurements for the EDA-system. The measurement points again represent the memory just after loading the report page. Here, no in-between measurements were taken, because no memory peaks were observed. Run 2 was a very long run and is therefore displayed in a separate table.

Run 1 (memory)		**Run 2** (memory)		**Run 3** (memory)		**Run 4** (memory)	
Area	memory (bytes)	Area	memory (bytes)	Area	memory (bytes)	Area	memory (bytes)
	3284080		3251080		3250064		3235216
100	7970940	20	4728876	63	4285748	34	3983160
	3476056		3607392		3587324		3616280
100	7990404	20	4746900	63	4300256	34	4000136
	3478980		3592276		3592408		3620680
100	7994152	20	4748220	63	4281344	34	4003640
	3484800		3617884		3623872		3625200
	3459060		3598004		3601648		3608568
	3490148		3665184		3634612		3616324
20	4784096	100	8180424	34	4016340	63	4308060
	3654872		3677316		3613420		3627024
20	4791084	100	8190880	34	4010880	63	4314012
	3659436		3681976		3642772		3632324
20	4796964	100	8195280	34	4025632	63	4321356
	3665136		3686672		3648960		3638476
	3649436		3658188		3626452		3622388
	3672848		3685208		3663644		3642312
63	4361220	34	4069652	100	8197584	20	4783700
	3680456		3693632		3694112		3653760
63	4367216	34	4075232	100	8207596	20	4790468
	3684524		3698244		3698992		3658304
63	4372852	34	4084832	100	8214076	20	4796624
	3689864		3707596		3704864		3664452
	3673016		3708972		3682792		3648628
	3700324		3712704		3710920		3692500
34	4084836	63	4402128	20	4850624	100	8216544

	3709408		3721404		3719496		3713432
34	4090828	63	4409244	20	4856688	100	8226572
	3713204		3726220		3724836		3716008
34	4096428	63	4414704	20	4862876	100	8231080
	3719136		3731840		3730692		3722432
	3721008		3733492		3708328		3712548

Table 28: Original system – memory measurement data

Run 1 (memory)		**Run 4** (memory)		**Run 5** (memory)	
Area	memory (bytes)	Area	memory (bytes)	Area	memory (bytes)
	60047488		60057192		60054404
20	60431180	100	60640496	34	60456444
34	60443032	63	60683572	20	60453820
100	60723704	20	60745628	100	60734532
63	60770632	100	61028032	34	60762788
		34	61055368	100	61042960
Run 3 (memory)		20	61118460	63	61089948
	60054312	34	61149948	20	61153576
100	60651188	63	61186224	63	61189824
34	60684460				
63	60712168	**Run 6** (memory)		**Run 7** (memory)	
20	60775776		60064608		60079676
34	60803064	63	60461416	63	60445536
100	61085528	100	60739216	20	60489824
20	61157324	34	60766700	63	60535216
63	61185224	63	60813548	100	60817404
34	61206976	20	60877368	34	60843092
100	61481792	34	60906708	20	60907320
20	61537740	20	60970688	34	60936276
100	61809492	100	61242444	100	61207404
63	61844476	63	61281840	63	61246824
34	61871932				
20	61918924				
34	61937560				
63	61976972				
100	62248672				

20	62304724		
63	62340248		

Table 29: EDA-system – memory measurement data, except run 2

Run 2 (memory)					
Area	memory (bytes)	memory (bytes)	memory (bytes)	memory (bytes)	memory (bytes)
	60048380				
20	60453724	64326500	68314456	72227088	76146380
34	60446312	64348276	68334484	72248824	76168136
100	60726780	64623112	68608968	72523660	76442992
63	60772872	64662492	68656424	72563040	76482392
20	60833888	64743132	68704764	72643380	76538416
34	60864444	64740344	68726200	72656896	76560172
100	61146656	65015108	69000868	72915328	76835028
63	61182940	65054444	69040612	72954916	76874344
20	61263420	65110512	69096780	73010512	76930412
34	61244580	65132288	69118556	73040804	76952148
100	61519172	65415172	69393372	73307552	77227036
63	61555580	65446504	69458204	73346932	77266436
20	61607424	65510752	69488816	73402944	77714544
34	61634048	65524220	69510272	73424528	77736300
100	61900860	65799056	69784960	73699296	78011108
63	61936440	65838436	69824348	73738676	77996208
20	61989256	65894520	69880400	73794760	78067424
34	62009800	65916256	69902156	73816516	60319036
100	62281424	66191092	70176972	74091332	60589992
63	62320804	66230024	70215924	74130712	60618196
20	62374184	66286556	70272136	74186828	60669000
34	62392276	66324696	70294212	74208512	60684228
100	62666988	66583128	70569028	74483348	60959056
63	62703232	66622540	70608428	74522728	60998144
20	62759224	66678552	70664172	74578580	
34	62781000	66700328	70685864	74600344	
100	63063056	66975196	70960660	74875140	
63	63094832	67014556	70998412	74914560	
20	63167280	67070660	71051328	74970604	

34	63172632	67092396	71072776	74992380
100	63447448	67367232	71347492	75267176
63	63486612	67406632	71386932	75326084
20	63560356	67462676	71442976	75379076
34	63564384	67484412	71472964	75384132
100	63839200	67759248	71777180	75658952
63	63878388	67800308	71778968	75698332
20	63934608	67932556	71835012	75764668
34	63956236	67978416	71856808	75776100
100	64255632	68228216	72131624	76050936
63	64270416	68267468	72171024	76090016

Table 30: EDA-system – memory measurement data, run 2