AALBORG UNIVERSITY
STUDENT REPORT

Master Thesis

Real-Time Embedded Systems in Java

Author:
Kristian Kolding Foged-Ladefoged

Supervisor:
Bent Thomsen

June 10, 2014

AALBORG UNIVERSITY
STUDENT REPORT

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg East
http://www.cs.aau.dk

Title: Real-Time Embedded Systems in Java

Theme: Programming Technology

Project Term:
10th semester, spring 2013

Project Group:
sw105f14

Students:
Kristian Kolding Foged-Ladefoged

Supervisor:
Bent Thomsen

Copies: 2

Pages: 47

Finished: June 10, 2014

Synopsis:

This project investigates the usefulness of object-oriented paradigm in real-time embedded systems. The investigation is twofold, with the benefits of the object-oriented paradigm explored through a literature review, and 28 benchmarks developed in Java to investigate whether or not the object-oriented paradigm has any influence on execution times.
The project concludes that the object-oriented paradigm promotes maintainability, reusability and increases productivity. Execution times are however significantly increased.

Summary

This project investigates the usefulness of object-oriented paradigm in real-time embedded systems. The investigation is twofold, with the benefits of the object-oriented paradigm explored through a literature review, and 28 benchmarks developed in Java to investigate whether or not the object-oriented paradigm has any influence on execution times.

The literature review is conducted based on a search strategy, which uses approaches suggested by Webster and Watson[32].

The benchmarks consists of 14 benchmarks, which is developed in two versions using object-oriented design and little to none object-oriented design. The benchmarks is derived from the well-known Malardalen benchmark suite[9].

The project concludes that the object-oriented paradigm promotes maintainability, reusability and increases productivity. Execution times are however significantly increased.

Table of contents

Chapter 1

Introduction

1.1 Motivation

Real-time embedded systems are a class of computer systems that monitor, respond, and interact with an external environment, such as the real world. The system interact with the environment through input and output interfaces such as sensors. The system is often required to compute and respond within a certain time frame, hence the name ”real-time”. It is essential for the system to finish within the time frame, exceeding the time frame can be devastating depending on the application. Real time embedded systems are ubiquitous and appear in a large variety of applications such as household appliances, multimedia, military, and medical[28].

Due to the time constraints of real-time embedded system, it is necessary to develop reliable and predictable software. This is achieved by developing a software architecture with limited overhead, such as unnecessary middleware. The language C is a popular language for developing real time embedded systems, due to its desirable characteristics such as the ability to access hardware.[22]

According to the article ”Struggle continues to plug embedded programming gap”[14], the American universities have shifted to teaching Java programming instead of C, for the reason that there are more jobs as Java developer than C developer. Unfortunately, this has created a generation of developers who are uneducated in C programming. Thereby created a lack of real-time embedded developers.

Java is a high-level object oriented programming language, that is often used as a general-purpose language. Java is often recognized as language with high portability[21], because it runs on a virtual machine. Java is viable language for real-time embedded system and is constantly receiving more support[22].

A study which observed bug and productivity rates for Java and C++ showed that C++ generates about 15 to 50 percent more defects and about 200 to 300 percent more bugs compared to Java. Furthermore did the study also show that Java is about 30 to 200 percent more productive that Java. The results may seem a bit skeptical, but regardless the study proved that Java is more productive and contains fewer bugs and defects[23].

1.2 Problem Definition

Java is high-level object-oriented programming language is very popular and has is increasingly gaining more support for real-time embedded systems. As described in section [**?**], it seems like the upcoming generation of software engineers lack the skills in real-time embedded system development due to the shift in programming language taught in the American universities.

To investigate the suitability of Java in real-time embedded system, I want to achieve insight in what benefits of the object-oriented paradigm. I have chosen to conduct a literature review to investigate the benefits of the object-oriented paradigm in regards to what the literature states. This brings me to my first research question which is:

"What benefits does the literature mention about the object-oriented paradigm"

Execution times is of high importance in a real-time embedded system, which is why I want to investigate the influence that the object-oriented paradigm may have on the execution times. This brings me to my second research question which is:

"Does the object-oriented paradigm influence the execution time of real time systems?"

Furthermore I want to achieve knowledge about real-time embedded systems and what makes Java a suitable language for real-time embedded systems.

Part 2 contains the search strategy used for my literature review, as well as the literature review itself. Part 3 contains knowledge about real-time embedded systems and what makes Java a possible candidate for real-time embedded development. Part 4 contains 24 benchmarks to investigate whether or not, the object-oriented paradigm has any influence on execution times.

Chapter 2

Literature Review

2.1 Search Strategy

Webster and Watson[**?**] describe an approach for high quality review. The approach focuses on structuring the review to make the research more complete. Complete reviews are defined as a review containing more than one research methodology, one journal, or one geographic location.

The approach consist of three components, identification of relevant and impacting articles, previously important articles, and new research bases on the previous articles. Webster and Watson suggest using leading journals related to the subject of interested, for searching for important and relevant articles. Based on these articles a backward and forward citation search should be conducted for identifying previously and subsequently important articles.

I have chosen to Web of Knowledge as search engine to search for scientific work. I find that Web of Knowledge best satisfies the requirements of Webster and Watson[**?**][25]. Web of Knowledge offers great options for constructing queries and limit the search to specific authors or publications. This helps narrowing the search for a more specific search result.

Constructing an initial search query can be difficulty due to the lack of knowledge for the search subject. I used an trial and error approach to find some initial articles. Based on these articles I constructed a search query which includes all articles describing object-oriented and level of abstraction, productivity, comprehension, bugs, defects, or quality attributes[13].

(OO OR ”object-oriented”) AND (”level* of abstraction” OR productivity OR compreh* OR bug* OR defect* OR maintainability OR modifiability OR analyzability OR testability OR readability OR reliability OR reusability)

The query returned 2677 articles in total, which had to be reduced to a more acceptable number of articles. Web of Knowledge provides the option to export various types of data about the search result, such as keywords for the articles. To reduce the amount of articles I used the keywords to analyze the search result. I developed a small program to count the number of times each keyword occurred. This analyze provided me with the insight that maintainability, modifiability, analyzability, testability, readability, reliability, and reusability were very popular keywords among the articles in the search result. Due to their popularity, I had to remove them.

The keyword program was further used to adjust the search query until the exported keywords reach a acceptable state of fitting and unfitting keywords. The final search query is as follows:

(OO OR ”object-oriented”) NEAR/1 (language OR paradigm OR design OR development) AND (defect* OR bug* OR productivity OR quality OR abstraction) The query returned 1282 articles in total. To reduce the amount of articles further more, the query was altered to only include articles from the top 100 most impacting journals in the field of computer science. The top 100 most impacting journals was identified using the impact factor tool from Web of Knowledge[24]. The journals was sorted based on the 5-year-impact attribute. The final search query was refined using the top 100 most impacting journals and returned 83 articles. The top 100 journals can be seen in appendix 6.1

The 83 articles was manual reduced by reading the title, abstract, and conclusion of each article. The manual-reduction narrowed the articles down to 20 articles in total.

Based on the 20 articles a backwards citation was conducted to identify the most important articles that the 20 articles cited. The backwards citation search was performed by utilizing Web of Knowledge option for exporting the list of citation of the 20 articles. Based on this list I picked the five most cited articles. Web of Knowledge also provides the option to perform forward citation search, that returns a list of all articles citing one of the 20 articles. Based on this list I picked the five most cited articles.

By using Webster and Watsons approach I found 30 articles in total. During the reading of the 30 articles, it was necessary to perform a final reduction of the articles due to realizing upon reading the articles, that some of the articles used object-oriented programming to achieve a goal but did not discuss why object-oriented programming was used to achieve the goal. The final number of articles is 11. A shortened version of the literature search can be seen in table2.1

Step	Articles	Description
1.	Query (1282)	Search query described above executed at `www.webofknowledge.com` – April 12th
2.	Top Journals (83)	Reduction by only including articles from top 100 journals.
3.	Manual reduction (20)	Narrowing results to articles that contain descriptions on the use of code metrics or quality metrics and preferably in combination with a project's characteristics.
4.	Backwards Citation (5)	From the citations from the articles in step 3, find the most cited article not already included.
5.	Forward Citation (5)	From the citations from the articles in step 3, find the most cited article not already included.
6.	Initial Result (30)	Combining articles identified in step 3 with articles identified in step 4 and 5.
7.	Final Result (11)	Removing irrelevant articles from step 6 upon reading.

Figure 2.1: Short description of the literature search.

2.2 Literature Review

The article **"A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs"**[16] points out that there is a lack of empirical evidence to support the benefits of the object-oriented paradigm.

The article goal is to provide empirical evidence that object-oriented design increases two important components of maintainability, which is understandability and modifiability.

The article compare the effect of design principles that are recognized as good and bad practice. The recognized good practices are as follow: Coupling between classes should be low, which is achieved by reducing the complexity and decreasing the number of messages between objects. Inheritance coupling should however be kept high, which is achieved by subclass being specialization of its generalization superclass. Cohesion should be kept high by letting classes carry out only one functionality. Services should only require a minimum of attributes such that no attributes is unused. Furthermore classes need to portray a specialization, and not some arbitrary choices of functionality. Lastly model names should closely correspond to the name of the concepts being modeled.

The article set up an empirical study were 33 computer science students would conduct maintainability on two different object-oriented systems, that were designed after the recognized good and bad object-oriented practice.

The study results showed that object-oriented systems developed following the good practice were significantly easier to maintain.

The article **”An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software”**[10] cites that a number of empirical studies have been performed to validate increased software maintainability using the object-oriented paradigm. However these empirical studies have been inconclusive and the majority of the studies have been conducted using students in controlled environments.

The articles goal is to perform a maintenance experiment on two versions of an operational real-world mission-critical system. The two versions are relatively developed in C using the structured paradigm and C++ using the object-oriented paradigm. Software professional were maintaining both systems.

The experiment showed that group members working with the object-oriented version required less effort to maintain to system. The article points out that the usefulness of UML for analysis may be a beneficial factor for the object-oriented paradigm and thereby making it superior to non object-oriented paradigms.

The article **”An Empirical Study of the Object-Oriented Paradigm and Software Reuse”**[26] investigates if the object-oriented paradigm improves reusability. The experiment described in the article, is based on a target system developed by a set of software engineering students. The students were divided into two groups, one group used the procedural paradigm and the other group used the object-oriented paradigm. The student groups were furthermore divided into two groups, one group using reuse and the other group not using reuse.

The experiment compared the four versions by the number of runs, run time errors, code edits, syntax errors, and the total time it took to fix run time errors. The experiment showed that the object-oriented paradigm improves productivity, although a significant part of the improvement is due to the effect of reuse. Reuse promotes productivity regardless of language paradigm, but the object-oriented paradigm is shown to promote the reuse process.

The article **”Assessing the cognitive consequences of the object-oriented approach A survey of empirical research on object-oriented design by individuals and teams”**[6] present a state-of-the-art review of the empirical research on object-oriented design. The article compares object-oriented design versus procedural design with the focus on design performed by individual and teams. Furthermore the article also focuses on reuse.

The review shows that novice object-oriented designers have difficulties in the process of class creation. The mapping between the problem domain and the programming domain is not easy. The notice object-oriented designers need to construct a representation of the procedural aspects of the solution in order to refine, evaluate and revise, in order to create proper classes.

The review supports that the object-oriented paradigm promotes reuse of software. The object-oriented paradigm is also shown to improve productivity, and that a significant part of the improvement is due to the effect of reuse.

The review also shows that the object-oriented paradigm helps overcome some problems encountered at the software design team level compared with traditional paradigms. Coordination and knowledge sharing is enhanced, and the communication between team members is more effective.

The article **”Cognitive Activities and Level of Abstraction in Procedural and Object-Oriented Design”**[18]observes experts in the object-oriented paradigm and novice object-oriented designers with procedural experience while they solve a design problem. Realism was of high importance for the study, so the developers used java development tools of their own choice.

The articles goal is to compare the two designs developed by the experts and novice object-oriented designers.

The article found that object-oriented designers decompose their designs into classes corresponding to the real world domain entities. The object-oriented designers also utilized reuse though built-in classes. Procedural designers decomposed their design according to actions on data structures.

The goal of **”Evaluating the effect of a delegated versus centralized**

control style on the maintainability of object-oriented software"[8] is to evaluate delegated and centralized control styles on the maintainability of object-oriented software.

Delegated control style is done by distributing a well-defined set of responsibilities among a number of classes. Each class have a specific role and critical for the overall system architecture. Centralized control style is where a few larger classes is selected as control classes. These control classes' responsibility is to coordinate the smaller and more simpler classes.

The article present a study with 99 junior, intermediate, and senior Java consultants. To compare the difference between professional developer and students, 59 undergraduate and graduate students also participated. The programming tasks consisted of six change tasks, a training task, a pretest task, and four incremental design tasks.

Two measurements were used to compare the delegated and centralized control style, the total effort to complete a task measured in time, and correctness of the implemented tasks.

The study showed that developers using delegated control style made a more elegant solution and a better object-oriented representation of the task to solve. However, the centralized control style offered better maintainability, due to only a few classes contained the majority of the application logic.

The article **"Identification of dynamic comprehension processes during large scale maintenance"**[2] states that during maintenance, software engineers must understand code for a variety of tasks, which requires a lot of effort and can be time consuming.

The article present an experiment, where a software engineer task is to maintain an emulation program of networking protocols. The software engineer was observed and recorded as he maintained the system. The goal was to see the comprehension process that the software engineer went through.

It was observed that the software engineers used a multilevel approach for obtaining knowledge about the code. The engineer frequently switched between program situations and domain models. Effective understanding of large-scale code needs substantial domain information.

In a related study, it was found that object-oriented experts uses a systematic top-down design strategy when they need to gain a greater understanding of a system. Whereas procedural experts uses a more opportunistic approach, in which they changes between levels of abstraction.

The article **"Identification of Move Method Refactoring Opportunities"**[19] propose a methodology for locating bad smells in object-oriented software. Some bad smells can be resolved by moving the source of the bad smell such as methods, to a more suitable location in the software architecture. The proposed methodology is able to evaluate whether or not the bad smell is reduced or resolved after the source has been moved.

The article mentions that low coupling and high cohesion reflects a good object-oriented design. The proposed methodology uses this knowledge together with coupling and cohesion metrics to determine whether or not to move a method.

By studying the coupling and cohesion metrics evolution on two open source projects, it has shown that refactoring performed based on the proposed methodology has a positive impact bad smells.

The article **"Predicting the probability of change in object-oriented systems"**[20] acknowledge of the importance of changes handling in the software development process. The goal of the article is to predict the probability of class changes in future of the software development. To predict the probability of class changes, the article present a methodology that uses three attributes, which is referred to as axes of change.

The inheritance axis represents changes in interfaces and class inheritances. The reference axis represent changes in class instantiations and declaration of methods. The dependency axis represent changes in class and package names.

The presented methodology for predicting changes in object-oriented system was proven to provide statistically insignificant predictions. The methodology has been automated and can be applied to any object-oriented software system for easy refactoring.

The article **"R++ Adding path-based rules to C++"**[7] mentions that one of the purposes of the object-oriented paradigm is to provide a higher level of abstraction, thereby developers do not have to deal with irrelevant implementation details and allows them to work at a more neutral level of problem solving.

As application become more complex, a lot of effort is used on the low level details of interobject dependencies. Maintaining these interobject dependencies is a non-trivial task, and can lead to errors of omission and logic by the developer maintaining the system.

The article adds path-based rules to the object-oriented programming language C++. The rules strictly follow interobject paths in a domain model as a new class member. The rules define automatic behavior of a class, and monitor data members for changes, which satisfy a rule's condition. If a rule is satisfied, its action will be executed.

Adding path-based rules to C++, has successfully provides a useful level of abstraction, allowing the developer to avoid dealing with low level implementation details when modelling dynamic collections of objects.

The article **"The class blueprint Visually supporting the understanding of classes"**[29] mentions that maintenance of software systems accounts for 50 to 75 percent of the overall cost of system development. Reading object-oriented code is more difficult than reading procedural code. The domain model of the application is distributed across the whole system, and is often modelled using inheritance and polymorphism, which decreases the readability of object-oriented code.

Understanding classes is of high importance in object-oriented systems and contributes to better maintainability in object-oriented systems. The article present a new approach, class blueprints, which is a visualization of semantically augmented call and access-graph of the methods and attributes of classes.

The article findings is that the suggested approach, class blueprints, reduces complexity by making assumptions about classes without having to read the whole source code. The blueprints helps to select the relevant methods whose reading is critical for understanding a class. A common vocabulary is defined by the blueprints, which eases the communication between developers. The blueprints helps the developers obtain a good understanding of classes with a minimal effort.

2.3 Literature Conclusion

This part has sought to answer my first research question, which is as follows:

"What benefits does the literature mention about the object-oriented paradigm"

The literature mentioned the lack empirical evidence that supports the claims that the object-oriented paradigm is more effective, productive, reuse, higher maintainability, and is less prone to defects and bugs. The lack of empirical evidence is also reflected in the number of articles found using the search strategy.

The literature did however validate some of the claims about the object-oriented paradigm. Object-oriented software systems developed using a good practice is requires less effort and are less time consuming to maintain, however using a delegated style is less beneficial than using a centralized style. The object-oriented paradigm was also showed to promote productivity, due to the high reusability provided by the object-oriented design. Object-oriented developers do not have to deal with irrelevant implementation details due to the high level of abstraction supported by the object-oriented paradigm. The object-oriented paradigm is also shown to help teams overcome problems during the design phase of software systems.

Overall does the object-oriented paradigm provide the developers with the required insight in the code to be productive and carry out maintainability with little effort.

Chapter 3

Real-Time Embedded Java

As the name suggests, real-time embedded systems, it is a combination of real-time systems and embedded systems. Embedded systems are system which are dedicated to solve a specific problem, where the coupling between hardware and software integration is tightly coupled. An embedded system is usually part of a larger embedded system, known as embedding system. An embedding systems consists of multiple embedded systems. Embedded systems can be applied to a large variation of application, such as security systems, telephones, televisions, and network routers. [15]

Real-time systems are computer systems that consists of a variety of tasks with time constraints, used to control, monitor, and react to an external environment, such as the real world. The system interact with the environment through input and output interfaces such as sensors.

The system has to comply accordingly to its time constraints, where time constraints are deadlines for a execution time for a given functionality. If these time constraints are not satisfied it could have a major impact on its environment.

Real-time systems can be applied to a large variation of applications, such as vehicle brakes, traffic control, communication, and household systems.[28]

3.1 Real-time system types

Real-time systems are categorized by three different types of time constraints depending on their application[15]. The three categories are hard, soft, and firm time constraints. The three types of time constraints are described below:

Hard real-time systems have no tolerance for missed deadlines. The computed results which are generated after a missed deadline is not useful for the system. Depending on the application, the penalty of a missed deadlines can be catastrophic and should always be avoided. Developing a hard real-time systems require great effort to predict whether or not the systems can uphold its time constraints. An example of a real-time system with hard deadlines could be the brake system in vehicles, where the time constraint is the delay between a person stepping on the brake until the brake activate. [15]

Soft real-time systems strives to meet its deadlines, but it a bit more flexible than hard real-time systems. Soft real-time systems can miss deadlines with none or only minor penalty. The computed results which are generated after the missed deadline are likely to still be useful for the system. A missed deadline can however delay the system and should therefore be avoided. An example of a real-time system with soft deadlines could be a DVD player. The DVD player decodes the video and audio streams while responding to user inputs. If a deadline is missed, the DVD player will respond slow to the user input, but the computed result will still be useful after the missed deadline. [15]

Firm real-time systems is viewed as a hybrid category between soft and hard real-time systems. Firm real-time systems is required to meet a certain amount of deadlines. The computed result which are generated after a missed deadline is not useful for the system and will be discarded. An example of a firm real-time system could be a mp3 player, where a missed deadline would lead to few missed bits and thereby decreases the sound quality. If too many deadlines are missed, the sound might just stop. [31]

As previously mentioned, real-time system consists of tasks, where each task is responsible for a specific functionality. There exists three types of tasks, sporadic, periodic, and aperiodic[5].

The tasks types are described as follows: **Periodic tasks** are repeatedly executed within a regular time interval, known as its period[5].

Sporadic tasks can arrive at the system at arbitrary points in time, but will only arrive with a minimum inter-arrival time between two consecutive invocations[5].

Aperiodic tasks are only invoked once and the arrival time of the task is unknown at design time. Aperiodic task cannot guarantee their nor other tasks' deadlines and are therefore not suitable for hard real-time systems[5].

3.2 Characteristics of real time embedded system

Developing a real-time system is a non-trivial tasks due to the several important characteristics of real-time systems. This section contains the most important characteristics.

Real-time systems interact with the real world using input and output interfaces such as sensors. The real-time system must compute responses accordingly to the input received from the real world. Real-time system must incorporate all possible inputs and events retrieved from the real world. Due to the real world being complex, the real-time systems often tend to be complex as well[30].

Depending on the application of the real-time system, the system should consist of high reliability, safety, and availability. The real-time system must be engineered to tolerate faults, such that the system can continue to operate regardless of whether a fault has occurred and the type of the fault[30].

Real-time systems interact with an external environment, which is achieved through devices such as sensors. The devices communicate with the processor through input and output registers. The devices may also sent an interrupt to signal the processor that a certain operations has been performed. Due to the time constraints of real-time systems, the interaction between the input and output ports, operations must be tightly coupled. Middleware should be avoided if possible.

Real-time systems require an efficient implementation such that it can meet its time constraints. This is the reason why imperative languages such as C are often prefered for developing real-time systems. C can interact with the hardware without a middleware layer and thereby making the implementation more efficient.

The memory footprint for real-time systems intended for a high number of produced units is very important. With good memory management it is possible to keep the memory footprint low and thereby keep production costs low as well. Due to real-time systems interacting with an dynamical environment, it may be necessary to store temporary data in dynamically allocated memory. The heap used for dynamically allocate memory is required to predictable, such that the system is schedulable.

Developing real-time system require tight coupling between software and hardware engineers. Furthermore significant effort is required for making a good requirements analysis to ensure the system reflects the desired needs before producing the required hardware. The requirements are often expressed formally as timed automata.[3]

Real-time program structure consists of three phases: initialisation, execution, and termination. Initialisation phase creates all the needed objects and threads throughout the systems life cycle. The initialisation phase will only be invoked once in the systems life cycle and it is not time-critical. In the execution phase, the tasks run concurrently and is therefore time-critical. The termination phase is invoked when all the tasks are completed. The termination cleans the system for old objects and threads and afterwards invokes the Initialisation phases and thereby a new life cycle begins.

3.3 Worst Case Execution Time

Worst case execution time(WCET) is the most important factor for real-time systems with hard deadlines. With the knowledge of each tasks' WCET and their corresponding task types it is possible to conduct a schedulability analysis and thereby make sure that the system is schedulable and no deadlines are missed.

Determining the WCET is a non trivial task and great effort is spent to improve or research new tools and methods to achieve greater precision. Precision is used to describe to which degree a derived WCET differs from the actual WCET of a task.

There exists a large variation of benchmarks suites for evaluating and comparing WCET methods and tools, such as the benchmark suite developed by Malardalen university in Sweden. The Malardalen benchmark suite consists of 35 programs developed in C. The programs consists of different algorithms and vary in size.

In 2006 Malardalen university announced a WCET challenge to evaluate the state-of-the-art in timing analysis for real-time systems and to encourage further research in the WCET community. The challenge assesses both academic and commercial WCET tools. This challenge also confirmed the importance of WCET analysis.

There exists different approaches for WCET analysis, I will briefly describe static methods and measure-based methods.

3.3.1 Static methods

The static methods approach attempts to provide WCET estimations by examining the an abstract representation of the code and combine it with an abstract representation of the desired system which the code is meant to be executed on. This approach provides safe WCETs, meaning that the actual WCET will always be lower than the estimated WCET. The abstract representation only represent one type of processor, the codes WCETs is therefore only guaranteed to be appropriate for this type of processor.

3.3.2 Measure-based methods

The measure-based methods approach measures the execution time of the code being executed on the desired hardware or in a simulation. The WCET is often more precise using measure-based methods than static methods. Measure-based methods are not desirable, even with the higher precision, due to the fact that the measure-based approach is very difficult to conduct. To measure the WCET using the measure-based approach, all the worst possible inputs must be identified which can be difficult.

3.4 Schedulability Analysis

The purpose of a schedulability analysis is to validate whether or not a real-time system can uphold all of its deadlines. There exists several schedulability algorithms, which ensures that real-time systems are reliable, two of those algorithms are fixed-priority scheduling and cyclic executive scheduling. These two algorithms are briefly explained below.

3.4.1 Fixed-Priority Scheduling

Fixed-priority scheduling(FPS) algorithm is widely adopted for real-time systems. In FPS, tasks are given a fix priority before the program is executed and thereby making FPS a static scheduling algorithm. FPS allow the usage of preemption and will always execute the task with the highest priority. FPS can easily be applied to both periodic and sporadic tasks, where sporadic tasks are treated as periodic with periods representing their minimum inter-arrival time.

Assigning priorities to the tasks can be dealt with using either of the following two approaches: Rate-monotonic priority assignment or deadline-monotonic priority ordering.

Rate-monotonic priority assignment assigns priorities such that the lower a task's period is, the higher priority the task will be assigned. Deadline-monotonic priority ordering assigns priorities such that the lower a task's deadline is, the higher priority the task will be assigned.

3.4.2 Cyclic Executive Scheduling

Cyclic executive scheduling(CES) algorithm consist of a fixed set of periodic tasks, that is determined before runtime and which makes CES a static scheduling algorithm. Each task is assigned into one or more procedures, which are called in a specific order. The order is constructed such that the deadlines of the tasks are upheld. The procedures are split into minor cycles. Each minor cycle starts periodically and runs for a fixed amount of time. The collection of all minor cycles needed to execute all tasks are referred to as a major cycle. [30]

CES is used because of its simplicity and easy implementation. However when a system reaches a certain size it becomes difficult to maintain. Changes in the procedures lead to changes in their WCET and thereby requires a reevaluation of the schedule.[30]

3.5 Safety-Critical Java

Java is a high-level object-oriented programming language and is relatively new in the real-time embedded community. Java is receiving significant interest in real-time use. Java required a real-time profile, for being used in real-time systems. Real-time profiles provide a specification as how properties of the program should be structured for being suitable for real-time.

There exists a variety of real-time profiles for Java, such as Safety Critical Java(SCJ)[1]. SCJ consists of an API and a set of rules as how to program real-time Java. Java is compiled into Java bytecode that runs typically runs on a Java Virtual Machine(JVM), which makes Java difficult to use in embedded systems. SCJ purpose is to sustaining and standardise safety critical systems developed in Java.

3.5.1 Mission

Applications developed using SCJ consists of one or more missions. A mission is defined as a set of periodic and aperiodic event handlers. Missions are assigned with a dedicated block of memory, referred to as mission memory.

Objects created in the mission memory persist until the mission is terminated and the mission resources will be released before the mission is terminated. All missions start in a initialization phase, where objects may be allocated in either the mission memory or the immortal memory.

When the initialization phase has completed, the mission enters a execution phase. During the execution phase, the mission may use and alter the created objects in either the mission or immortal memory. All processing for missions occurs in one or more schedulable objects.

When a schedulable objects is released it enters its initial scoped memory. The scoped memory is not shared with other schedulable objects. When a mission receives termination request, all of its objects are notified to stop operating, such that the mission can safely stop and run clean-up before being the mission is terminated.

3.5.2 Compliance Levels

SCJ provides three compliance levels to accommodate the large variety of safety-critical applications. Safety-critical application can be very different and vary greatly in complexity, such as single-thread applications and multi-threaded applications. The three compliance levels are described as follows:

Level 0 applications is a model often described as a cyclic executive model. Using this level, the mission can be thought of a set of computations, which is executed in a periodically clock driven timeline and repeatedly executed throughout the missions lifetime. Level 0 schedulable objects are only suitable for periodic event handlers, which consists of a period, deadline, and a start time. All periodic event handlers are executed with the control of a single thread, which is why there is no synchronisation concerns. Level 0 applications can create private memory, but it cannot share them with any other periodic event handlers.

Level 1 applications consists of a single mission sequence with a set of concurrent computations, each with a priority controlled by a fixed-priority preemptive scheduler. Level 1 application can consists of periodic and aperiodic event handlers. Objects are shared in the mission and immortal memory among the applications periodic and aperiodic event handlers using synchronized methods to maintain the integrity of the objects.

Level 2 applications starts with a single mission, but is allowed to create and execute additional missions. Level 2 missions consists of periodic, aperiodic event handlers, and no-heap real-time threads. No-heap real-time threads, are real-time threads that do not have access to the heap, and can therefore continue to run even during a garbage collection cycle.

3.6 Real-Time Java Platforms

In this section, I examine two platforms, namely Java Optimized Processor(JOP)[27] and Hardware near Virtual Machine(HVM). The JOP is a processor which is designed to execute Java bytecode and is time predictable. The HVM is a virtual machine, implemented in C, which interprets Java bytecode. HVM is available for several embedded hardware platforms.[12].

JOP The JOP is designed for time-predictable execution of real-time tasks developed in Java. JOP is an implementation of the Java Virtual Machine in the hardware, such that Java bytecode can be executed directly on the processor. JOP is intended for applications in embedded real-time systems and the primary implementation is on a field programmable gate array, with RAM, storage, and input/output ports. In general purpose processors the instructions are written is machine code, whereas instructions run on the JOP is Java bytecode. This means that there is no need of middleware to interpret nor translate the Java bytecode into machine code.

The safety critical java profile is been implemented for the JOP. In addition to the JOP, there also exists a JOP emulator, which simulates the JOP. The emulator offers support for debugging, such that the code does not have to be executed on the actual JOP.

HVM The HVM is lean Java virtual machine for real-time embedded systems[12]. The virtual machine is written in C and interprets Java bytecode compiled by a standard Java compiler. The HVM run on bare bone, meaning that an real-time operating systems is not needed. The HVM supports an increasing variety of embedded hardware platforms such as: Arduino Uno, EV3, Atmel ATmega 2560[12]. The HVM allow execution natively on a computer without the use of an emulator.

The HVM consists of two things, a plug-in for the Eclipse IDE and a API with all the necessary methods for developing real-time applications for the HVM. SCJ compliance level 0 and 1 are implemented in the HVM, thereby providing the ability to schedule periodic and aperiodic tasks.

The HVM supports hardware objects [12] which allows developers to interact with the input and output hardware ports using native Java. The hardware objects increases the level of abstraction, which is good according to the literature review in chapter 2.

Developers can choose to either encapsulate the hardware objects, which would increase the level of abstraction, or can operate on the input and output ports using bitwise operators, which however may not increase the level of abstraction but may be more suitable for smaller applications. HVM also supports native C code to be called from within the Java program. The desired native C methods must be declared in a separate C file and referred to in the Java program.

Chapter 4

Java Benchmarks

This chapter is dedicated to answer the research question: "'Does the object-oriented paradigm influence the execution time of real time systems?" To answer this research question I have reengineered 14 of the well-known Malardalen benchmark programs in two different versions, both written in Java. One version using object-oriented design and the other version with none or little use of object-oriented design.. Object-oriented metrics have been applied to the benchmarks to validate the degree of object-oriented design used in the benchmarks.

4.1 Object-oriented metrics

In my previous semester project I researched metrics that reflects quality in object-oriented systems. Based on this research I found out that metrics are characterized by four distinct characteristics: complexity, cohesion, coupling, and inheritance.

Complexity describes the interaction between entities in a software systems. Complexity is always desired to be low, which promotes good maintainability, modifiability, analysability, and testability.

Cohesion describes to which degree methods and attributes belongs together. High cohesion supports good reliability and reusability.

Coupling describes the dependencies among the classes and methods. Low coupling is prefered and supports good readability and maintainability.

Inheritance describes the amount of inheritance used in a software system. High inheritance provides good reusability and maintainability.

Using metrics that covers the four characteristics, provides insight and degree of object-oriented design that is applied to the benchmarks. The popular and widely accepted C&K[4] metric suite covers the four characteristics, which is way I have chosen this metric suite for measuring the benchmarks.

C&K metric suite consists of six metrics, namely: weighted methods per class(WMC), depth of inheritance tree(DIT), number of children(NOC), coupling between classes(CBO), response for class(RFC), and lack of cohesion of methods(LCOM). In addition to the C&K metric suite, I have decided to also measure the code size. The size metric used is lines of code(LOC)

In my previous semester project, I discovered that literature describing and validating metrics tend to have their own interpretation of metrics, even well-known metrics such the C&K metrics. I will therefore provide a brief description of each metric and how they are calculated.

WMC - Computes the sum of each method complexity in a class. This metric often rely on a complexity metric to calculate the complexity of each method. Due to the benchmarks programs being fairly small in size, I have decided to simplify this metric, such that all methods have the complexity value one. The WMC value is prefered to be low.

DIT - Computes the total of superclasses for a given class using a tree structure. All my benchmarks are written in a single file, meaning that nested classes will be interpreted as a subclass to a superclass. Therefore will the DIT value represent the number of classes in additional to the main class. The DIT value is prefered to be low.

NOC - Computes the number of direct descendants for a given class. The NOC value is prefered to be low.

CBO - Computes the number of coupling between two classes. A coupling exists when two classes shares at least one method class, field access, inheritance, argument, return type, object declaration, or exception. The CBO value is prefered to be low

RFC - Computers the total number of method declarations and method calls in a class. The RFC value is prefered to be low.

LCOM - Computes the total number of method that are implemented in the proper classes. LCOM decides if a method is implemented propperly based on the number of pairs of methods in the class which share one or more instance variables. The LCOM value is prefered to be high.

LOC - Computes the total number of lines of the java bytecode rather than the total number of lines in the source code. The reason for this is because line of source code does not reflect the size of a program, because multiple statements can be represented on one line. The LOC value has no preferable states since it only reflect the size of the program.

For simplicity, I have decided to not show the metrics for each class but rather for each benchmark. This decision is based on the fact that the programs are relative small and only consists of one or two classes. Furthermore this decision provides a better overview of the metrics for all the programs.

4.2 Java benchmark

Because of difference in the language design of C and Java, it was not possible to make a objective translation into Java, without the influence of my personal coding style. The chosen Malardalen benchmarks are selected based on related work, where the 14 benchmarks were translated into Java for the purpose of validating a WCET tool for Java processor[17].

The related work inspired me to develop two versions of the same 14 benchmarks, one using object-oriented design(OO) and the other using little to none object-oriented design(NOO). The two versions were developed such that a comparison of the execution time could be made. The benchmarks are implemented similar in both versions, it is only the program structure that is different. Both programs utilizes the same facilities provided by java, such that there is only difference is type nor methods used.

The selected benchmarks are briefly described below:

Binary Search program does a binary search for the array of 15 integer elements.

Bubble Sort program tests the basic loop constructors with integer comparison and simple array handling. The program sorts 100 integers.

Cyclic Redundancy Check program demonstrates a cyclic redundancy check operation on 40 bytes of data. The program contains complex loops and a lots of decisions.

Exponential Integral program computes an exponential integral function. The program contain simple loops and bitwise operations.

Fast Discrete Cosine Transform program performs a discrete cosine transform on a two dimensional array, where both dimension lengths are

8. The program contains a lot of calculations based on integral array of elements.

Fibonacci program calculates a 30 digit sequence of fibonacci numbers. The program only contains a two nested loops.

Insertion Sort program performs a simple insertion sort for an array of 11 integers. The program contains one nested loop.

Janne Complex program contains two nested loops with a lot of integer comparison.

Matrix Count program counts the sum of non-negative numbers in a matrix. The program consists of a lot of nested loops.

Matrix Multiplication program multiplies two square matrices with size of 20x20. The program consists a lot of multidimensional array handling.

Nested Search program performs a search on a 4 dimensional array. The program contains four nested loops.

Quick Sort program performs a quick sort algorithm on an array of 20 integer elements. The program contains three nested loops.

Select Number program selects the n^{th} smallest number in an array. The program consists of three nested loops and a lot of integer comparison.

Lower Upper Decomposition program simulates a linear equation by lower upper decomposition. The program contains a lot of nested loops, integer comparison, and array handling.

Table 4.1 displays the values of the metrics applied to the OO version of the program. Table 4.2 displays the values of the metrics applied to the NOO version of the program.

When dealing with program of small sizes it is hard to determine the quality of the programs, and in general higher WMC, DIT, NOC, CBO, RFC, and LCOM values indicates increased use of object-oriented design, but does not reflect whether it is good or bad design.

The tables show that WMC values of the OO version are higher than the NOO version, this would normally conclude that the NOO version is of higher quality, but due to the small size of both program versions, it may indicates that the OO version in in fact more object-oriented than the NOO version due to the increase of methods used.

The tables shows that the DIT values for the NOO version is all equal 0, reflecting no use of extra classes. The DIT values are however either 0 or 1, which indicates that some programs are utilizing extra classes.

The NOC values are equal zero in both program versions. This shows that neither of the versions uses any class inheritance.

The CBO values for the NOO version is all equal zero as expected since that version does not contain any class objects. The CBO values for the OO versions displays some coupling between objects in the program that uses extra classes.

The tables show that the RFC values are higher in the programs utilizing extra classes. The RFC values for the OO versions with no utilization of extra classes is higher than the NOO version which confirms that the OO versions is more object-oriented than the NOO version.

The LCOM values are significantly higher in the OO version than the NOO version, which again confirms that the OO version is infact more object-oriented.

The size of both versions are about the same, regardless of the amount of object-oriented design applied to the programs.

In total the metrics shows that the use of object-oriented design is higher in the OO version than the NOO version, which was expected.

	WMC	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**	**LOC**
BinarySearch	6	1	0	2	11	10	234
Bubble	7	1	0	2	11	15	121
Crc	9	0	0	0	12	36	483
ExpInt	5	0	0	0	8	10	168
FDCT	5	1	0	2	10	6	1064
Fibonacci	4	0	0	0	7	6	48
InsertionSort	6	1	0	2	10	10	143
JanneComplex	4	0	0	0	7	6	52
MatrixCount	6	0	0	0	9	15	117
MatrixMult	6	0	0	0	9	15	137
NestedSearch	8	1	0	2	14	8	6057
QuickSort	5	0	0	0	8	10	365
Select	5	0	0	0	8	10	289
SLE	5	0	0	0	8	10	

Table 4.1: This table shows the values of the applied metrics to the OO benchmark version

	WMC	DIT	NOC	CBO	RFC	LCOM	LOC
BinarySearch	3	0	0	0	6	3	255
Bubble	3	0	0	0	6	3	81
Crc	5	0	0	0	8	10	458
ExpInt	3	0	0	0	6	3	157
FDCT	3	0	0	0	6	3	970
Fibonacci	3	0	0	0	6	3	43
InsertionSort	3	0	0	0	6	3	105
JanneComplex	3	0	0	0	6	3	50
MatrixCount	3	0	0	0	6	3	102
MatrixMult	4	0	0	0	7	6	123
NestedSearch	3	0	0	0	6	3	6036
QuickSort	4	0	0	0	7	6	360
Select	4	0	0	0	7	6	282
SLE	3	0	0	0	6	3	286

Table 4.2: This table shows the values of the applied metrics to the NOO benchmark version

4.3 Benchmark results

The execution time analysis have been conducted on both the HVM and JOP platform. The reason for selecting both platforms for execution time analysis, is to outrule that one platform may favor either object-oriented or non object-oriented design.

I chose SymRT as tool for calculating the execution times. SymRT has higher precision[11] than other state-of-the-art execution time tools in real-time Java. Furthermore, the SymRT tool supports both the HVM and JOP which is preferable. SymRT generates a network of timed automata, which can be executed in UPPAAL to calculate both the best and worst execution time. I have decided to use both worst and best case execution time, such that the both execution times can be compared accordingly to the OO and NOO version of the programs. The best and worst case execution times are both measured in cycles.

Table 4.3 shows the best and worst case execution time, of both benchmark versions running the on HVM platform. Table 4.4 shows the best and worst case execution time, of both benchmark versions running the on JOP platform.

By comparing the BCET of both benchmark versions on the HVM platform, it shows that the BCET is significantly higher when using object-oriented design. The WCET is also significantly higher when using object-oriented design. The JOP platform shows the same pattern as the HVM platform, that the BCET and WCET is significantly higher when using object-oriented design.

The JOP benchmark contains an unexplainable results from the nested search NOO version. The benchmark programs for the HVM and JOP platform are the same, however it is only the JOP platform that contains an unexplainable result.

	BCET NOO	BCET OO	WCET NOO	WCET OO
BinarySearch	977	24231	1081	42339
Bubble	1677	2934	1883	4231
Crc	7179	9252	11748	27306
ExpInt	1648	2112	2518	3610
FDCT	1617	14139	1825	23953
Fibonacci	1179	1643	1740	2730
InsertionSort	900	1997	1004	3193
JanneComplex	880	1411	985	2079
MatrixCount	1721	2869	1927	3998
MatrixMult	3794	4310	4819	5693
NestedSearch	818	1995	923	3691
QuickSort	704	1338	807	2007
Select	704	1168	807	1797
SLE	3121	4483	3528	5853

Table 4.3: This table shows the benchmark results of both version using the HVM as platform

	BCET NOO	BCET OO	WCET NOO	WCET OO
BinarySearch	921	6238	921	6238
Bubble	65	335	65	335
Crc	2328	2656	2514	4922
ExpInt	72	288	94	307
FDCT	1203	5887	1203	5887
Fibonacci	45	176	52	182
InsertionSort	233	651	233	651
JanneComplex	39	165	39	165
MatrixCount	103	349	103	349
MatrixMult	475	517	475	517
NestedSearch	19653	436	19653	436
QuickSort	414	773	414	773
Select	405	714	405	714
SLE	205	664	205	664

Table 4.4: This table shows the benchmark results of both version using the JOP as platform

4.4 Benchmark conclusion

This part of the report has sought to answer my second research question, which is as follows:

"Does the object-oriented paradigm influence the execution time of real time systems?"

To answer the research question I have developed 12 Java benchmarks in two different versions, one using object-oriented design and one using little to none object-oriented design. The benchmarks have been tested on two platforms, namely JOP and HVM, to show that the chosen platform did not favor neither version of the benchmark.

To show that the two benchmark versions are using different levels of object-oriented design, I applied the well-known C&K metric suite of the following six metrics: WMC, DIT, NOC, CBO, RFC, and LCOM.

The benchmark result shows that the object-oriented paradigm does in fact influence both WCET and BCET, regardless of the platform. The BCET is increased by up to 2480 percent and the WCET is increased by up to 3916 percent.

The benchmarks are developed by a single developer, and may therefore be influenced by the developer subjective opinion about proper object-oriented design. The benchmarks need to be evaluated by multiple developers to validate the applied object-oriented design.

Chapter 5

Conclusion

In this project, the study regulation required that the project should be on a current research problem in computer science community and the project should reflect understanding of the research problem. I chose to investigate the possibilities of using Java in real-time embedded systems. I narrowed my research into be about the benefits of using the object-oriented paradigm for real-time embedded systems, which led me to the following two research questions:

"What benefits does the literature mention about the object-oriented paradigm"

"Does the object-oriented paradigm influence the execution time of real time systems?"

My first research questioned was answered by conducting a literature review using a search strategy based on the approaches suggested by Webster and Watson[**?**]. The yield of the search strategy was low numbered, which is understandable after conducting the literature review. A lot of the literature mentioned the lack of empirical evidence supporting the claims the benefits of the object-oriented paradigm.

The literature did however support some of the claims, showing that the object-oriented paradigm support maintainability, reuse, increases productivity, and is less defect and bug prone.

My second research question was answered by developing 14 benchmarks in two different version, one using object-oriented design and using little or none object-oriented design. To make sure that a singular platform favored either of the benchmark version, the benchmarks was conducted on two different platforms, namely JOP and HVM. To validate to which degree the object-oriented design was applied to both benchmark version, the well-known C&K metric suite was applied. The C&K metric suite confirmed that the two benchmark versions was developed with different degree of object-oriented design.

The BCET and WCET was calculated using SymRT, which can generate a network of timed-automata. Using UPPAAL, it was possible to calculate the WCET and BCET based on these network of timed-automata. The benchmark results showed that the BCET and WCET was dramatically increased using object-oriented design on both platforms.

The developed benchmarks need to be further validated, to ensure that the implementation is correct and the object-oriented design is applied properly.

Object-oriented Java promotes some great features, but the execution time does unfortunately suffer due to the object-oriented paradigm. Object-oriented Java may do well in large real-time embedded systems, where hardware costs are of low importance. Java developed using little or none object-oriented design may be suitable for smaller real-time embedded systems or systems where hardware costs is of high importance such as in mass production.

Chapter 6

Appendix

6.1 Top 100 Journals

Journals have been sorted in alfabetical order.

ID	Journal Title
1	ACM Transactions on Applied Perception
2	ACM Transactions on Autonomous and Adaptive Systems
3	ACM Transactions on Computer-Human Interaction
4	ACM TRANSACTIONS ON DATABASE SYSTEMS
5	ACM Transactions on Embedded Computing Systems
6	ACM TRANSACTIONS ON GRAPHICS
7	ACM TRANSACTIONS ON INFORMATION SYSTEMS
8	ACM Transactions on Information and System Security
9	ACM Transactions on Internet Technology
10	ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE
11	ACM Transactions on Multimedia Computing Communications and Applications
12	ACM Transactions on Sensor Networks
13	ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY

14	ACM Transactions on the Web
15	Ad Hoc Networks
16	ADVANCES IN ENGINEERING SOFTWARE
17	ANNUAL REVIEW OF INFORMATION SCIENCE AND TECHNOLOGY
18	COMMUNICATIONS OF THE ACM
19	COMPUTER-AIDED DESIGN
20	COMPUTER COMMUNICATIONS
21	COMPUTER COMMUNICATION REVIEW
22	COMPUTER GRAPHICS FORUM
23	Computer Networks
24	COMPUTERS & SECURITY
25	COMPUTER STANDARDS & INTERFACES
26	COMPUTER
27	DATA & KNOWLEDGE ENGINEERING
28	DATA MINING AND KNOWLEDGE DISCOVERY
29	DECISION SUPPORT SYSTEMS
30	Electronic Commerce Research and Applications
31	EMPIRICAL SOFTWARE ENGINEERING
32	Enterprise Information Systems
33	EUROPEAN JOURNAL OF INFORMATION SYSTEMS
34	GEOINFORMATICA
35	GRAPHICAL MODELS
36	IBM JOURNAL OF RESEARCH AND DEVELOPMENT
37	IEEE Communications Surveys and Tutorials
38	IEEE COMPUTER GRAPHICS AND APPLICATIONS
39	IEEE INTERNET COMPUTING
40	IEEE MICRO
41	IEEE MULTIMEDIA
42	IEEE NETWORK
43	IEEE PERVASIVE COMPUTING
44	IEEE SOFTWARE
45	IEEE Systems Journal
46	IEEE Transactions on Computational Intelligence and AI in Games
47	IEEE Transactions on Dependable and Secure Computing

48	IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE
49	IEEE TRANSACTIONS ON INFORMATION THEORY
50	IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING
51	IEEE TRANSACTIONS ON MOBILE COMPUTING
52	IEEE TRANSACTIONS ON MULTIMEDIA
53	IEEE TRANSACTIONS ON RELIABILITY
54	IEEE TRANSACTIONS ON SOFTWARE ENGINEERING
55	IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS
56	IEEE WIRELESS COMMUNICATIONS
57	IMAGE AND VISION COMPUTING
58	INFORMATION & MANAGEMENT
59	INFORMATION PROCESSING & MANAGEMENT
60	INFORMATION SCIENCES
61	INFORMATION AND SOFTWARE TECHNOLOGY
62	INFORMATION SYSTEMS
63	INFORMATION SYSTEMS FRONTIERS
64	INTERNATIONAL JOURNAL OF ELECTRONIC COMMERCE
65	INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE
66	INTERNATIONAL JOURNAL OF MEDICAL INFORMATICS
67	International Journal on Semantic Web and Information Systems
68	Internet Research
69	JOURNAL OF THE ACM
70	JOURNAL OF THE AMERICAN MEDICAL INFORMATICS ASSOCIATION
71	JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE AND TECHNOLOGY
72	Journal of Ambient Intelligence and Smart Environments

73	Journal of the Association for Information Systems
74	Journal of Chemical Information and Modeling
75	Journal of Cheminformatics
76	JOURNAL OF INFORMATION SCIENCE
77	JOURNAL OF INFORMATION TECHNOLOGY
78	JOURNAL OF MANAGEMENT INFORMATION SYSTEMS
79	JOURNAL OF MATHEMATICAL IMAGING AND VISION
80	JOURNAL OF NETWORK AND COMPUTER APPLICATIONS
81	Journal of Optical Communications and Networking
82	JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION-RESEARCH AND PRACTICE
83	JOURNAL OF STRATEGIC INFORMATION SYSTEMS
84	JOURNAL OF SYSTEMS AND SOFTWARE
85	JOURNAL OF VISUAL COMMUNICATION AND IMAGE REPRESENTATION
86	Journal of Web Semantics
87	MATHEMATICAL AND COMPUTER MODELLING
88	MATHEMATICAL PROGRAMMING
89	METHODS OF INFORMATION IN MEDICINE
90	MIS QUARTERLY
91	MOBILE NETWORKS & APPLICATIONS
92	ONLINE INFORMATION REVIEW
93	Personal and Ubiquitous Computing
94	SIAM Journal on Imaging Sciences
95	SIMULATION MODELLING PRACTICE AND THEORY
96	Software and Systems Modeling
97	SOFTWARE TESTING VERIFICATION & RELIABILITY
98	VLDB JOURNAL
99	WIRELESS COMMUNICATIONS & MOBILE COMPUTING
100	WORLD WIDE WEB-INTERNET AND WEB INFORMATION SYSTEMS

Bibliography

[1] Safety critical specification for java. special communication with jsr 302 group, 2010.

[2] A.M. Vans A. von Mayrhauser. Identification of dynamic comprehension processes during large scale maintenance. 1996.

[3] Hans Toetenel Bas Graaf, Marco Lormans. Embedded software engineering: The state of the practice. 2003.

[4] S R Chidamber and C F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 1994.

[5] Gerhard Fohler Damir Isovid. Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. 2000.

[6] Francoise Detienne. Assessing the cognitive consequences of the object-oriented approach a survey of empirical research on object-oriented design by individuals and teams. 1997.

[7] Anil Mishra Diane Litman, Peter F. Patel-Schneider. R++ adding path-based rules to c++. 2002.

[8] Dag I.K. Sjoeberg Erik Arisholm. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented softwaree. 2004.

[9] Andreas Ermedahl Jan Gustafsson, Adam Betts. The mälardalen wcet benchmark: Past, present and future. 2010.

[10] Stephen R. Schach Joa Sang Lim, Seung Ryul Jeong. An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software. 2005.

[11] Bent Thomsen Kasper Soee Luckow, Corina S. Pasareanu. Symbolic execution and model checking for timing analysis of java real-time systems. *Aalborg University*, 2014.

[12] Stephan Korsholm. Hvm lean java for small devices.

[13] Rasmus Hoppe Nesgaard Aaen Kristian Kolding Foged-Ladefoged, Kasper Møller Andersen. Software quality - what code metrics can tell us. 2014.

[14] George Leopold. Struggle continues to plug embedded programming gap. `http://www.eetimes.com/document.asp?doc_id=1261676`, 2012. Last visited 15/5/2014.

[15] Qing Li. *Real-Time Concepts for Embedded Systems*. CMP Books, 2003.

[16] John W. Daly Lionel C. Briand, Christian Bunse. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. 2001.

[17] Benedikt Huber Martin Schoeberl, Rasmus Ulslev. Worst-case execution time analysis for a java processor. *Software Practice and Experiment*, 2009.

[18] Bob Rehder Nancy Pennington, Adrienne Y. Lee. Cognitive activities and level of abstraction in procedural and object-oriented design. 1995.

[19] Alexander Chatzigeorgiou Nikalaos Tsantalis. Identification of move method refactoring opportunities. 2009.

[20] George Stephanides Nikalaos Tsantalis, Alexander Chatzigeorgiou. Predicting the probability of change in object-oriented systems. 2005.

[21] Oracle. The java language environment. `http://www.oracle.com/technetwork/java/neutral-137138.html`. Last visited 5/4/2014.

[22] Oracle. The education of embedded systems software engineers: failures and fixes. `http://www.embedded.com/design/programming-languages-and-tools/4238223/The-education-of-embedded-systems-software-engineers--failures-and-fixes`, 2012. Last visited 5/4/2014.

[23] Geoffrey Philips. Embedded software engineering: The state ofthe practice. 1999.

[24] Thomsen Reuters. Web of knowledge impact factor tool. `http://webofknowledge.com/JCR`, 2014.

[25] Thomson Reuters. Why use web of knowledge. `wokinfo.com/about/whatitis/`, 2013. Last visited 15/12/2013.

[26] Dennis G. Kafura Sallie M. Henry, John A. Lewis. An empirical study of the object-oriented paradigm and software reuse. 1991.

[27] Martin Schoeberl. Jop: A java optimized processor for embedded real-time systems. 2005.

[28] Alan Shaw. *Real-Time Systems and Software*. Wiley, 2001.

[29] Michele Lanza Stephane Ducasse. The class blueprint visually supporting the understanding of classes. 2005.

[30] Michele Lanza Stephane Ducasse. Real-time systems and programming languages: Ada 95, real-time java, and real-time posix. 2009.

[31] Scott Brandt Tim Kaldewey, Caixue Lin. Firm real-time processing in an integrated real-time system. 2006.

[32] Jane Webster and Richard T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Q.*, 2002.