Middle East Technical University
Department of Computer Engineering

CENG 491 – COMPUTER ENGINEERING DESIGN

REQUIREMENT ANALYSIS REPORT

“KROMPILER”

BY

CORESOFT

Osman Paşalak
Fatih Akdoğan
Alperen Yeşil
M. Ozan Koloğlu

Fall 2007
# TABLE OF CONTENTS

1. INTRODUCTION ................................. HATA! YER İŞARETİ TANIMLANMAMIŞ.
   1.1 PROJECT TITLE ................................ .............................. 3
   1.2 PROJECT DEFINITION .......................................................... 3
   1.3 PROJECT SCOPE .............................................................. 3

2. TEAM ORGANIZATION .............................. 5
   2.1 TEAM MEMBERS ............................................................... 5

3. MARKET RESEARCH ................................. 6
   3.1 LITERATURE SURVEY ...................................................... 13

4. SYSTEM REQUIREMENTS ............................. 16
   4.1 FUNCTIONAL REQUIREMENTS ............................................. 16
      4.1.1 Software Requirements .................................................. 16
      4.1.2 Hardware Requirements .................................................. 17
   4.2 NON-FUNCTIONAL REQUIREMENTS ..................................... 17
      4.2.1 Performance ................................................................. 17
      4.2.2 Usability ................................................................. 17
   4.3 USER REQUIREMENTS ...................................................... 18
      4.3.1 Use Case Diagram ......................................................... 18

5. PROJECT SCHEDULE ........................................... 20

6. RISK MANAGEMENT ........................................... 20
   WE EXPLAIN OUR RISK PLAN IN KROMPILER BELOW. ...................... 20

7. MODELING ................................................... 24
   7.1 DATA MODEL ................................................................. 24
   7.2 FUNCTIONAL MODEL ......................................................... 24
      7.2.1 Data flow Diagrams ......................................................... 24
      7.2.2 Explanation of DFD ......................................................... 26
   7.3 PROCESS MODEL .............................................................. 27

8. APPENDIX .................................................. 27
   A -GANTT CHART ................................................................. 27
1. Introduction

1.1 PROJECT TITLE

Our project title is ‘Krompiler’

1.2 PROJECT DEFINITION

As you know many new hardware architectures are designed each year, they all need compilers. That is why compiler technology is very important. It is for sure that compiler will produce the right output but moreover it is very important for a compiler to use less amount of time. Today many developers are interested in optimizing compilers. Handling these optimizations is very significant. Our aim in this project is to do optimizations for CSTAR compiler framework, to write a code generator for the optimizations and to make an optimization manager.

1.3 PROJECT SCOPE

Krompiler project consists of mainly 3 parts which are optimizations, test case generator, and optimization manager. Here are some general features that will be in these parts.

OPTIMIZATIONS

Optimization is the process of modifying a system to make some aspect of it work more efficiently or use fewer resources which can be hardware specific or hardware independent. For example using reduced amount of memory stack, cpu cycles are kinds of optimizations.

Optimizations are the most important part of the compiler.

Here are the optimizations we will write in our Krompiler project

1) Constant folding
2) Dead code elimination
3) Basic block ordering
4) Local/Global forward substitution
5) Strength reduction
6) Unreachable code elimination
7) Dead object elimination
8) Local/Global Copy Propagation
9) Local/Global Common Subexpression Elimination
10) Jump Optimizations
11) If Simplifications
12) Tail merging
13) Loop unrolling
14) Loop reversal
15) Loop splitting / peeling
16) Loop fission (distribution )
17) Loop fusion (combine)
18) Loop switching (interchange)
19) Loop unswitching
20) Loop skewing

**TEST CASE GENERATOR**

Testing is one of the most important parts of a software project. For a compiler it is not acceptable to generate a wrong code. So we should test the compiler with a set of tests. Since it may not be convenient to use hand-written tests. So it is better to use a test case generator. In our project we will implement a test case generator which will be helpful especially for the optimizations. The developer will be able to generate a test code as he/she wants. He can generate a test code for just one optimization or more. He/she also will be able to choose the ratio of the operations. (e.g. %20 multiplication, %10 if conditions) By the help of the test case generator developer will have good test-codes and this will better help to see the bugs in the system.
OPTIMIZATION MANAGER

Many compiler optimizations are actually transformations that compiler writers hope that it would optimize the input. There are few optimizations that always result in better code. Therefore a sequence of optimizations are good for some input sources and bad in some other. So it is hard for a compiler to choose what to do. In this project we will write an optimization manager which will help compiler users to apply optimizations as many times and in any order. Optimization manager will read an external file which specifies the execution order of the optimizations and each optimization’s options. (e.g. number of times that the optimization will be applied)

2. Team Organization

Our team has Democratic Decentralized (DD) structure because of extensive technical design issues and also framework that we add our classes is so huge. Project contain variety of different parts. Every team member is given duties with a part and responsible for own part. In weekly briefings we debate about each other topics and make some suggestions to each others

Task partitioning among team members is explained in Gantt chart. Different colors are used for different tasks to identify task division.

2.1 Team Members

Team members and their contact mails are:

Osman Paşalak  queueuc@gmail.com
Fatih Akdoğan  e1394592@metu.edu.tr
3. Market Research

In our market research we look for same type of compiler frameworks. Here is a list of compiler frameworks that we found and inspected.

**COSY Compiler Development System**

CoSy is flexible, easy-targetable compiler development system developed by ACE Associated Compiler Experts. With its modular design and extensive use of software generators, CoSy is one of the world’s best environment for professional construction of production quality compilers for the broad range of processor architectures. Many large and small semiconductor industries have successfully used CoSy in the development of their C/C++ compilers. While CoSy has always been available for selected research in advanced compilation techniques, an increasing number of researchers recognize the advantages of working with this well-supported and maintained compiler framework.

**Building a Cosy Compiler**

Building a Cosy compiler can be outlined by the following description.

**STEP1:** As the first step in the construction of a CoSy compiler, the target description file (TDF) is created. The TDF contains detailed information on the target architecture's data model, the sizes of the C data types and the available memory types.
STEP2: Other processor architecture related information is specified in the CoSy code generator description file (CGD). As a trivial starter in the CGD, the register file and characteristics of available registers are described.

STEP3: A more elaborate task within the CGD is to map the CoSy intermediate representation (IR) to the instructions that are available in the architecture's instruction set. This mapping by so-called RULES basically encompasses the core of the code generator. Starting from the available prototype code generator and guided by Compiler Trainer’s step-by-step incremental testing, this is a straightforward job.

STEP4: In the last stage towards a validating compiler, the compiler and the compiler supervisor that CoSy generates need to be configured. As for the aforementioned CGD, the so-called compiler configuration is best based on one of the available prototype EDL files in CoSy. This configuration defines which algorithms ("engines") should be included in the compiler. Some of these algorithms require additional configuration, e.g. to specify the stack frame layout, spill frames and parameter passing conventions.

STEP5: With the focus on creating an optimizing compiler that takes full benefit of the architecture's features, instruction scheduling is the first optimization that could be taken up. The CGD is augmented with a SCHEDULER section that contains two sets of descriptions. One description set focuses on latencies, specifying the number of cycles it takes for the result of an instruction to become available for another instruction. The other set describes resource use, specifying the cycles in which the architecture's functional units are occupied by an instruction. Furthermore, the rules that exhibit non-default latencies or resource use are annotated with the defined latency classes and resource use patterns.

NEXT STEPS: After creation of the initial, validating compiler, the next step is to add specific features, such as Dwarf debug information support, inline assembly, multiple calling conventions, special pragma support, as well as further optimization of the compiler for best performance. Depending on the requirements of the compiler,
additional effort in obtaining the desired performance may be necessary. This is where CoSy users once again experience the ease of enhancing a CoSy compiler by adding specific analysis and optimization engines. CoSy's robust, extensible framework really enables compiler engineers to focus at the main objective: a high quality, high performance compiler.

Who Use Cosy Compiler

- **Semiconductor companies**
  for fast and cost-effective development of production-quality compilers for new processor architectures
- **Development tool and EDA companies**
  for development of commercial software development tools
- **Architecture R&D groups**
  for performance testing and architecture roadmap exploration
- **Academia and Research Institutions**
  for research on compilation techniques and processor architecture design
ICD-C Compiler Framework

ICD-C is a High-Level Framework that handles the target application at Source Code Level. It is used by ICD/ES as a compiler frontend, responsible for parsing the source code and performing High-Level optimizations.

In ICD-C, all high-level information is retained, even including whitespace and comments. This feature makes ICD-C the optimal choice for the implementation of Source-to-Source optimizations in a stand-alone prepass optimizer. Customers can thus keep their environments and add performance-improving prepass optimizations as pluginmodules. Customers wishing to develop their own compilers or tools in-house can obtain a library or even source-code license of ICD-C.

Properties Of The ICD-C Compiler Framework

- Input language support:
  - Full C99 compliant, including C99 parser
  - Optional deactivation of undesired C99 features
- High level intermediate representation:
- Preservation of entire C code structure, even including comments
- High-level to mid-level transformation module

- Support for stand-alone Source-to-Source optimizations:
  - Facility for generating compilable C code
  - Allows pre-pass optimizations in a given toolchain

- Analyses:
  - Control flow, data flow, function callgraph analysis
  - Automatic maintenance of consistency

- Multiple Compilation Units
  - Allows really global optimizations
  - Global analysis of functions and symbols

- C++ interface:
  - Full access to ICD-C's data structures
  - Allows code analysis, manipulation and optimzation

- Backend interface:
  - Built-in Interface for code generation using ICD-CG
  - Compatible to the popular 'olive' code generator generator
  - Interface to ICD's compiler backend ICD-LLIR

Possible Applications of the ICD-C Framework

- Stand-alone pre-pass source code optimizations
  - Plug-Ins for existing compiler toolchains
  - No modification of existing frameworks required
  - Exploit the high optimization potential at the source level

- Front-End for setting up a complete compiler toolchain
  - Full control over your in-house compiler development
  - Benefit from the consulting services offered by ICD/ES

- Source Code analysis features
• Check code for conformance to design guidelines
• Implement proprietary 'lint' applications
  • C code obfuscators
    • Protect your software's IP by a pre-pass code obfuscation
    • Code functionality and performance is maintained

EDG(Edison Design Group)

Edison Design Group is a corporation dedicated to developing and licensing compiler front ends. Currently, we offer front ends for C/C++, Java™ and Fortran 77.
The C++ Front End

The front end does complete syntax and semantic analysis, including complete error checking. It produces about 1,600 different error messages. Diagnostics always display the source line with a caret indicating the exact position of the error. The level of diagnostic output can be controlled in various ways.

The front end translates source programs into a high-level, tree-structured, in-memory intermediate language. Implicit and overloaded operations in the source program are made explicit in the intermediate language, but constructs are not otherwise added, removed, or reordered. The intermediate language is not machine dependent (e.g., it does not specify registers or dictate the layout of stack frames). The front end can optionally generate raw cross-reference information, which can be used as a basis for building source browsing tools.

The front end includes an integrated preprocessor, which can do modern or pcc-style preprocessing. Normally, the preprocessor runs as part of the front end, and no intermediate textual file is created. However, a preprocessed output file can be produced if desired. Precompiled header files can be created and used. Also included: a C-generating back end, which can be used to generate C code for C++ programs; a C++-generating back end, which is useful for source-to-source transformation applications; a prelinker, which handles automatic template instantiation; a minimal runtime library (but not any "real" libraries, e.g., for stream I/O; see our list of library vendors); utilities to write the intermediate language to a file, read it back in, and display it in human-readable form; and a name demangler.

The Customers

Customers are companies that want to develop a compiler or a source-analysis tool, companies such as computer manufacturers, chip manufacturers, and software
tool developers. They license front ends from the company, combine them with software of their own (e.g., a code generator), and sell the resulting products.

3.1 Literature Survey

Our project scope contains a lot of optimizations and it will be useful to state clearly what these optimizations are. Our aim to put explanations about optimizations is to prevent misunderstood in our project. Optimizations manager and test case generator is explained enough in scope part. We thought that there is no need further technical explanation.

Here explanations of optimizations that we will implement in Krompiler.

- **Constant Folding**
  Constant Folding is the process of simplifying constant expressions at compile time. Terms in constant expressions are typically simple literals, such as the integer 2, but can also be variables whose values are never modified, or variables explicitly marked as constant

- **Basic Block Ordering**
  Basic block ordering is to order blocks to reduce the frequency of jumps.

- **Dead Code Elimination**
  Dead code elimination removes instructions, statements, and expressions which are not used.

- **Local/Global Forward Substitution**

  In local forward substitution, right hand side of assignments in to objects is forward substituted in to other statements which uses the object. It is effective only within basic blocks. It can also be used to reduce register pressure. It is also known as inverse common subexpression elimination. Global forward substitution is effective with in a function.
• **Strength Reduction**
  Strength reduction is an optimization where a function of some changing variable is calculated more efficiently by using previous values of the function.

• **Unreachable Code Elimination**
  Unreachable code elimination removes basic blocks that cannot be reached from entry basic block. Such codes can be present in the source code due to infinite loops.

• **Dead Object Elimination**
  Dead object elimination removes unused local objects and globals having static linkage.

• **Local/Global Copy Propagation**
  Copy Propagation replaces copies of variable with the original name, eliminating redundant copies.

• **Local/Global Common Subexpression Elimination**
  Replaces multiple instances of an expression with a temporary variable holding its result but may have an side effects for it generates new temporary variables which may need to be stored in memory.

• **Jump Optimizations**
  Jump optimization simplifies jumps to the following instruction, jumps across jumps, and jumps to jumps. It deletes unreferenced labels and unreachable code, except that unreachable code that contains a loop is not recognized as unreachable.

• **If Simplifications**
  These optimizations can reduce the number of conditional branches in a program. If simplification apply to if constructs with constant-valued conditions and remove the arm of the if which is unexecutable.

• **Tail Merging**
  Tail merging searches for basic blocks in which the last few instructions are identical and that continue execution at the same location either by one branching to the instruction following the other or both branching to the same location. What the
optimization does is to replace the matching instructions of one of the blocks by a branch to the corresponding point in the other.

Loop constructs in a program are the regions where payoffs are greatest. So additionally we will implement some loop optimizations.

- **Loop Unrolling**
  
  Loop unrolling is the process of expanding a loop so each new iteration contains several of what used to be an iteration. It reduces loop overheads such as index variable maintenance and control hazards in pipelines. It also allows better instruction scheduling. 
  But it has a disadvantage that while expanding a loop it increases the code size.

- **Loop Reversal**
  
  Loop reversal reverses the order in which values are assigned to the index variable. This is an optimization which can help eliminate dependencies and thus enable other optimizations.

- **Loop Splitting/Peeling**
  
  Loop splitting attempts to simplify a loop or eliminate dependencies by breaking it into multiple loops which have the same bodies but iterate over different contiguous portions of the index range. A useful special case is loop peeling which can simplify a loop with a problematic first iteration by performing that iteration separately before entering the loop.

- **Loop Fission (Distribution)**
  
  Loop fission attempts to break a loop into multiple loops over the same index range but each taking only a part of the loop's body. This can improve locality of reference both of the data being accessed in the loop and the code in the loop's body.

- **Loop Fusion (Combine)**
  
  Loop fusion reduces loop overhead. When two adjacent loops would iterate the same number of times their bodies can be combined as long as they make no reference to each other's data.

- **Loop Switching (Interchange)**
  
  Loop interchange is the process of exchanging the order of two iteration variables. It improve the cache performance for accessing array elements.
• **Loop Unswitching**
  
  Loop unswitching moves a conditional inside a loop outside of it by duplicating the loop's body, and placing a version of it inside each of the if and else clauses of the conditional. This can improve the parallelization of the loop, but it doubles the amount of the code generated.

• **Loop Skewing**
  
  Loop skewing takes a nested loop iterating over a multidimensional array, where each iteration of the inner loop depends on previous iterations, and rearranges its array accesses so that the only dependencies are between iterations of the outer loop.

---

**4. System Requirements**

4.1 Functional Requirements

4.1.1 Software Requirements

1) **LINUX**: QUICKC compiler framework works under Linux. However under Windows Cygwin and a Linux virtual machine environment, the framework should also work.

2) **GCC**: QUICKC compiler framework is build with GCC version 4. So, GCC version should be over 4.0. Developers also need GCC 4 to generate executable from framework assembly output. In our project we will write some of the optimizations for the QUICKC compiler. Because all the framework is written in C++, we have to work in GCC version 4, also.
4.1.2 Hardware Requirements

1) **X86 HOST MACHINE:** The compiler framework is designed for X86 machines. To execute the framework and to see the test results a X86 Host Machine is needed.

4.2 Non-functional Requirements

4.2.1 Performance

In a compiler framework, compiler optimizations consists extensive number of cases to handle as a matter of fact design of these optimizations effects compilers performance very much. Our aim is to increase performance as much as possible. For us, CSTAR provide two benchmark programs (Dhrystone, Whetstone). We will use these programs for enough time to inspect an optimal performance.

4.2.2 Usability

In a compiler framework, testing a produced compiler is an important issue and we will inspect this in test case generator part. Test codes for compilers should be as rich as possible and contain all functionality of input language. Krompiler will contain a complex test case generator. Also this generator will get options in human-readable form and produce highly complicated test code.
4.3 User Requirements

4.3.1 Use Case Diagram

Figure below shows user cases of Krompiler.
4.3.2 User scenario
Type of user for Krompiler will be a compiler engineer. Compiler engineer will use our classes and other utilities to design his/her compiler. Krompiler will provide Optimizations, Test Case Generator and Optimization manager for the user.

Optimizations for compilers can be used via anatrop manager object. Our optimizations will be registered to anatrop manager singleton object and they can be called with wanted options object and by name. There will be 20 optimizations for usage.

Test case generator is essential in compiler design for reducing future errors. Compiler engineer can use test code generator with different option set like 20 percent loops, 30 functions minimum etc. This generator also will be used by anatrop manager object because it is a derived class from anatrop.

Optimization manager is a practical tool for compiler testing and debugging. It is also an anatrop in the framework and can be used like other parts of Krompiler. Compiler engineer will use manager by filling a human-readable configuration file which is constructed by simple loops and conditions for choosing target code optimizations. Optimization rules automatically read by manager and executed as wanted.

### 5. Project Schedule

Gantt chart of the project Krompiler is in Appendix A. Also it contains information about task division.

### 6. Risk Management

We explain our risk plan in Krompiler below.
Aim

In a software project like ours, there are always potential risks that can negatively affect the normal flow of the development process. Identifying these risks and managing them is a must for that project to be successful. As Murphy mentioned in his laws: “If anything can go wrong, it will…”. Of course this is not the case all the time but the extra effort to handle these risks before they happen is extremely less than the effort to handle the problems after these risks come true. So the thing we will do here is handling the risks by identifying and managing them. In order to do this we have prepared a risk plan to have an efficient and more controllable process.

Risk Analysis

Identification

- **Unity of the team.** Since we expect the project to finish in two semesters - relatively long time for a student - one or more team member may leave the team with mandatory or urgent or even arbitrary reasons.

- **Time problem.** Since this is our first group project, we may not easily get used to group work, nevertheless, we have different lectures and different schedule. Coming together may not be easily possible all the time. Also since we have other lectures that have homework and projects; time management could be a problem.

- **Size estimate.** Size estimate may be significantly low.

- **Data loosing.** Loosing data by any kind of disc or system crashes.

- **Customer needs & liking.** We don’t have any irritating customers, so our customers are our instructor and the company we are working with. We think we will get along with them.

- **Lack of knowledge about development tools.** Since we are not familiar with the QuickC framework, learning its complete features may take some time.
• **Lack of technology support.** Our supporting company is located in the Netherlands. So, our main support method is e-mail. We will not be able to speak with our supervisor face to face. This can raise some problems, especially about time.

• **Common bugs in development environment.** We have a framework which is developed very recently. Although our supervisor seems quite experienced, the software is not guaranteed to be bug-free. It is possible that the tools can have some fatal bugs which will terminate or delay our project.

• **Security.** If we can not secure our code, our project may encounter some problems like thievery or sabotage during the process. But since our project is not a web or network application, this risk is low with respect to the others.

**Risk Table**

<table>
<thead>
<tr>
<th>RISKS</th>
<th>CATEGORY</th>
<th>PROBABILITY</th>
<th>IMPACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>D. Env. Risk &amp; Staff Risk</td>
<td>%5</td>
<td>3</td>
</tr>
<tr>
<td>Unity of the team</td>
<td>Business Impact</td>
<td>%10</td>
<td>1</td>
</tr>
<tr>
<td>Time problem</td>
<td>Employee Risk</td>
<td>%25</td>
<td>3</td>
</tr>
<tr>
<td>Loosing data</td>
<td>General risk</td>
<td>%10</td>
<td>2</td>
</tr>
<tr>
<td>Size estimate</td>
<td>Project size risk</td>
<td>%30</td>
<td>2</td>
</tr>
<tr>
<td>Customer needs &amp; liking</td>
<td>Customer risk</td>
<td>%20</td>
<td>3</td>
</tr>
<tr>
<td>Lack of knowledge about development tools</td>
<td>Develop. Env. Risk</td>
<td>%25</td>
<td>2</td>
</tr>
<tr>
<td>Lack of technology support</td>
<td>Technology risk</td>
<td>%15</td>
<td>2</td>
</tr>
<tr>
<td>Common bugs in development environment</td>
<td>Develop. Env. Risk</td>
<td>%10</td>
<td>1</td>
</tr>
</tbody>
</table>
Impact values:

1 – Catastrophic
2 – Critical
3 – Marginal
4 – Negligible

OUR METHODS TO HANDLE THESE RISKS

**Unity of the team.** We had an agreement about this risk. None of us will withdraw from this course and nobody will leave this project without a mandatory reason (health problems etc.). In such a case this person will talk about his leave with our instructor.

**Time problem.** All the group members have showed their schedule and we arranged a few meeting times per week to come together. Also most of the courses we take are same.

**Loosing data.** If we loose our data during the process, this will be a major hazard for our project. As we see this is a critical risk. Because of this, all the data we have produced will be copied separately to all of group members’ computers.

**Size estimate.** We divided our schedule into time slices and made a Gannt chart. If we have failed to finish one of these tasks in time, we will have overtime works to catch our schedule.

**Customer needs & liking.** We will consult with our instructor and supervisor regularly so if we encounter a conflict or an undesired situation, we will quickly handle that situation so that no problem will remain.

**Lack of knowledge about development tools.** We have already read most of the documentation that our supervisor has supplied to us. For unclear points we will read the rest of the documentation and begin practicing on the framework. We will start coding as soon as possible. Thus, while making progress on the project, we will clarify and strengthen our knowledge about the tools we use.
Lack of technology support. We have enough tech. support for now but as we proceed, we can have technical problems. Just after we see any such problems we will rescan the documentation for the solution. If we cannot find any answers, we will quickly communicate with the supervisor via e-mail.

Common bugs in development environment. It is possible that our tools have some bugs; so we will try to search and find out these bugs before we encounter them in a bad situation. When we find any bugs in the framework, we will make our supervisor know immediately. We will clearly explain characteristics of the bug. We will intervene if we can and help our supervisor about the problem, otherwise we will stand by until the bug is fixed.

Security. We will take necessary and usual precautions against thievery and sabotage of our code when we copy or store. If something happens, we will continue our work with our backup data. Since this is a course project it will not be practical to seek and defend our legal rights on the courts.

7. Modeling

7.1 Data Model

Krompiler doesn’t contain any relational databases. There is a storage of configurations for optimizations manager via a file. As a result we don’t show any database diagram (Entity Relationship diagram) for Krompiler.

7.2 Functional Model

7.2.1 Data flow Diagrams
Data Flow Diagram Level 0

Data Flow Diagram Level 1
7.2.2 Explanation of DFD
7.3 Process Model

In our project we will use evolutionary model as the process model. The main reason for selecting this process model is struggling with algorithms and architecture more than other parts. Also this is the reason why we eliminate other process models.

We have good understood requirements in this project and main characteristic of our project is optimal algorithm design. We have flexibility in our job and openness to new methods in implementation.

In implementation we will start optimizations which are well known algorithms but after this part, we will implement test case generator and optimization manager which are highly flexible in implementing. As a result evolutionary process model is most suitable model rather than other models.

Also time managing problems in this process model can be solved by adapting strict timing plan.

8. Appendix

A - Gantt Chart

Chart Legend : 

- All Members
- Osman & Fatih
- Alperen & Ozan
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Team Gathering and Determining Project Topic</td>
<td>29/09/2007</td>
<td>01/10/2007</td>
<td>4w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Determining Project Topic Details and Proposal</td>
<td>29/09/2007</td>
<td>05/10/2007</td>
<td>2w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Requirement Analysis</td>
<td>05/10/2007</td>
<td>06/10/2007</td>
<td>6w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Literature Survey and Market Research</td>
<td>09/10/2007</td>
<td>19/10/2007</td>
<td>8w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Risk Analysis</td>
<td>16/10/2007</td>
<td>19/10/2007</td>
<td>6w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Project Scheduling</td>
<td>15/10/2007</td>
<td>22/10/2007</td>
<td>4w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Project Quality Plan</td>
<td>22/10/2007</td>
<td>25/10/2007</td>
<td>3w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Analysis Report</td>
<td>29/10/2007</td>
<td>02/11/2007</td>
<td>4w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Milestone</td>
<td>02/11/2007</td>
<td>02/11/2007</td>
<td>2w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Practice Framework</td>
<td>02/11/2007</td>
<td>05/11/2007</td>
<td>2w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Design Discussion</td>
<td>05/11/2007</td>
<td>06/11/2007</td>
<td>8w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Information Gathering about Optimizations Test Case Generator and Optimization Manager</td>
<td>05/11/2007</td>
<td>19/10/2008</td>
<td>1w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Test Case Generator Design</td>
<td>13/11/2007</td>
<td>20/12/2007</td>
<td>2w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Optimization Manager Design</td>
<td>20/12/2007</td>
<td>26/12/2007</td>
<td>1w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Initial Design Report</td>
<td>29/12/2007</td>
<td>30/12/2007</td>
<td>1w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Prototype Implementation</td>
<td>30/12/2007</td>
<td>19/01/2008</td>
<td>7w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Detailed Design</td>
<td>30/12/2007</td>
<td>14/01/2008</td>
<td>2w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Final Design Report</td>
<td>03/01/2008</td>
<td>11/01/2008</td>
<td>4w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Project Prototype and Demo</td>
<td>16/01/2008</td>
<td>18/01/2008</td>
<td>6w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>