LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
School of Business and Management
Degree Program in Computer Science

SOUTH URAL STATE UNIVERSITY
Faculty of Computational Mathematics and Informatics
Degree Program in Fundamental informatics and information technology

*Mariia Kibel*

# A SOFTWARE ENGINEERING APPROACH TO DISTRIBUTED SYSTEM FOR COMPOSITE MODELING

1st Supervisor/Examiner: Prof. Ahmed Seffah, LUT

2nd Supervisor/Examiner: Cand. Sci. Natalja Dolganina, SUSU

Lappeenranta – Chelyabinsk

2015

# ABSTRACT


| | |
|---|---|
| **Author:** | Kibel Mariia |
| **Title:** | A software engineering approach to distributed systems for composite modeling |
| **Department:** | LUT School of Business and Management<br>SUSU Faculty of Computational Mathematics and Informatics |
| **Master's Programme:** | Computer science |
| **Year:** | 2015 |
| **Master's thesis:** | Lappeenranta university of technology,<br>South Ural State University, 50 pages, 6 tables, 17 figures |
| **Examiners:** | Prof. Ahmed Seffah<br>Cand. Sci. Natalja Dolganina |
| **Keywords:** | Distributed systems, modeling, composites, finite elements |



The goal of this thesis is to define and validate a software engineering approach for the development of a distributed system for the modeling of composite materials, based on the analysis of various existing software development methods. We reviewed the main features of: (1) software engineering methodologies; (2) distributed system characteristics and their effect on software development; (3) composite materials modeling activities and the requirements for the software development. Using the design science as a research methodology, the distributed system for creating models of composite materials is created and evaluated. Empirical experiments which we conducted showed good convergence of modeled and real processes. During the study, we paid attention to the matter of complexity and importance of distributed system and a deep understanding of modern software engineering methods and tools.

## РЕЗЮМЕ


| | |
|---|---|
| **Автор:** | Кибель Мария |
| **Заглавие:** | Программно-инженерный подход к распределённой системе для моделирования композитов |
| **Факультет:** | ЛТУ Факультет Безнеса и Менеджмента<br>ЮУрГУ Факультет Вычислительной Математики и Информатики |
| **Магистратура:** | Фундаметальная информатика и информационные технологии |
| **Год:** | 2015 |
| **Диссертация:** | Лаппеенрантский Технологический Университет,<br>Южно-Уральский Государственный Университет, 50 страниц, 7 таблиц, 16 рисунков |
| **Экзаменаторы:** | Кандидат техн. наук Наталья Долганина |
| **Ключевые слова:** | Распределённые системы, моделирование, композиты, конечные элементы |



Целью данной работы является определение и проверка программно-инженерного подхода к разработке распределённых систем для моделирования композитных материалов, основанные на анализе различных существующих методов программной инженерии. В ходе работы были рассмотрены основные особенности: (1) программно-инженерных методологий; (2) характеристик распределённых систем и их влияние на программную разработку; (3) моделирования композитных материалов и требований для разработки программного обеспечения. Применяя дизайн-ориентированный подход в качестве исследовательской методологии, распределённая система для создания моделей композитных материалов была разработана и оценена. Проведённые эмпирические эксперименты показали хорошее схождение между смоделированными и реальными процессами. В ходе исследования было уделено внимание вопросам сложности и важности распределённых систем, а также глубокому пониманию современных методов и инструментов программной инженерии.

## ACKNOWLEDGEMENTS

This thesis is the final stage of very intense year of my Double Degree program. It was very challenging but still I could say it was the best year in my life. I'm very grateful for this opportunity to LUT.

And my participation wouldn't be possible if I didn't apply for master's program in my home university, SUSU, at the first place. So I'm thankful to my supervisor, Natalja Dolganina for the inspiration and help. It's been a great pleasure working with you.

I want to say special thanks to all amazing people who've been close to me. Especially to my good friends, Sveta and Alejandro. We've had the greatest time together. And thank you for all your help and support. And of course, to my dearest Tanya, for being with me from the very beginning of this journey.

And I want to express deep gratitude to Outi and Liisa who've been there for me in my darkest times. Thank you for all your help and your endless kindness.

Mariia Kibel
Chelyabinsk, Russia 06.11.2015

# TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

## LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| CAD | Computer-aided design |
| CAE | Computer-aided engineering |
| FE | Finite element(s) |
| SOA | Service-Oriented Architecture |

# 1 INTRODUCTION

Nowadays high-performance fibers based on fibrous polymers have become a major material used for the ballistic protection due to their beneficial characteristics such as light weight, great performance against impact, high-temperature resistance, and ability to be formed in any shape [1, 2]. In the creation of armor plates as upper layers are widely used and as rear layers used fabric or non-woven (felt) materials.

Developing new composite materials and products made of them is a complex multistage process. It requires technical knowledge about their properties, qualities, and dependence on characteristics of the structure of these materials. This knowledge could be gathered from experiments. Many analytical and experimental studies of ballistic impact against textile composites were conducted. However, real-life experiments are too expensive in terms of time and materials consuming. Therefore, engineers usually use computational modeling for conducting ballistic experiments. One of the problems of this approach is the model's creation for further processing.

This problem could be solved with the help of models generating software. For each type of models, for example, ceramic, felt, and glass fiber composites different approaches should be used. It makes models preparations more time-consuming due to the need of various instruments usage.

In case of ceramic materials modeling starts from the graphical model, which is difficult to import from CAD tools to CAE tools for calculations due to complex geometry of the model (a large number of curved surfaces). Not all surfaces are imported and some of them deformed. The method of three-dimensional reconstruction allows creating finite element mesh for computation without using CAD tools.

In case of fibrous materials: felt and glass fiber composites the models have to be created from scratch, according to defined properties of model and threads.

## 1.1 Research objectives and questions

The goal of this thesis is to study software engineering approach to distributed systems and to apply it to composite modeling system. The main focus is finding the best solution for implementation this system. For this purpose, finished development will be evaluated.

Therefore, the main research question could be stated as follows:

”How to apply software engineering to distributed system for scientific modeling?”

Which could be decomposed into the following sub-questions:

1. What are the key principles and approaches of software engineering as a disciple of engineering for the development of complex software systems?
2. What are the key characteristics of a distributed system, how do they affect software development?
3. What are the challenges of composite modeling and how do they affect software development of a system for composite modeling?

## 1.2 Research process

The thesis will use design science research methodology which allows constructing and evaluating artifacts to meet organizational needs. It is important in disciplines aimed at creating successful artifacts [27]. Design science research includes seven major principles [22]:

1. **Design as an Artifact** which means that viable artifact should be produced as a result of research;

2. **Problem relevance**: the aim of design research is to develop solutions to important problems;

3. **Design evaluation** includes demonstration of the design artifact utility, quality, and efficacy;

4. **Research contributions** should be provided by design research in clear and verifiable form;

5. **Research rigor:** construction and evaluation of the design artifact should be conducted by employment of rigorous methods;

6. **Design as a search process -** the finding of efficient artifact requires using available means to reach established goals with respect to the rules of the problem environment;

7. **Communication of research** involves an effective presentation of design science research and its results.

Therefore, according to these principles the research will start from the study of the literature about development methodology. User requirements will establish objectives of a solution. Studied method will be the base for the system design and development. Finished system will be tested and evaluated.

## 1.3 Structure of the thesis

The second chapter takes a look at iterative software engineering approach for distributed systems. It covers distributed system definition and features, its major quality attributes such as interoperability, scalability, and usability. Also, this chapter takes a look at software development life circle of distributed systems. It shortly introduces software process, in general, then examines iterative software engineering approach and developing methodology for distributed systems. This information can be used for building a solution for modeling system.

The third chapter takes a look at the design and implementation of the composite modeling

system. Starting with introducing subject area and description of modeled materials, covers system requirements, functional and non-functional, describes user interface prototyping, following by design and analysis, and finishing with the implementation of the system.

The fourth chapter describes conducted testing and evaluation to assure the quality of the developed system.

The final chapter draws the conclusions of the research. It presents the main findings of the research.

# 2 ITERATIVE SOFTWARE ENGINEERING FOR DISTRIBUTED SYSTEM

The goal of this literature review section is introducing to the reader concept of distributed systems. First, it includes definition to provide an understanding of the topic main idea. Secondly, the main reasons of growing importance of distributed systems are reviewed. And, finally, going deeper in the subject, distribution criteria are reviewed.

Next focus of this chapter is one of the major quality attributes of distributed systems: interoperability, scalability, and usability. The meaning of these terms and possible dimensions and characteristics are studied to gain an understanding of how to implement them in developed system.

Finally, this chapter reviews software development lifecycle of distributed systems. It serves to analyze software engineering approach and plan developing methodology. In the beginning, the basics of software process and its major activities are presented. After that, a closer look was taken at iterative software engineering approach and its overall flow. Presented characteristics of iterative development help to justify the choice of that methodology. And, after that software process in the context of distributed systems is reviewed to adjust chosen approach to studied system.

## 2.1 Distributed systems

Nowadays distributed systems are gaining great importance. The following section presents this topic. It starts with the introduction of distributed systems definition. Then, it reviews the reasons of growing popularity of distributed systems and considers its advantages and disadvantages. For the purpose of gaining an understanding on distributed system development, the major distributed architecture models were studied and main development challenges were reviewed.

### 2.1.1 Definition

According to Tannenbaum [27], a distributed system is a collection of independent computers that appears to its users as a single coherent system. Distributed computing is a model in which components of a software system are shared among multiple computers to improve efficiency and performance. It is an integration of system services, presenting a transparent view of multiple computer systems with distributed resources and controls as a single computer system.

Term "distributed" was originally used to emphasize physical distribution within some geographical area. However nowadays it is used in a much wider sense, sometimes referring to logical distribution of the processes within one physical computer [6].

Over the past years, distributed systems have gained significant importance. The major advantages and the main reasons of their growing importance are the following [5, 25]:

- **Geographically distributed environment.**
  Since there is a need for speeding up the computation, it is possible to retrieve more computational power be the use of multiple processors. By dividing a problem into set of subproblems and assigning them to autonomous processors that can operate concurrently, it is possible to enhance the computation speed. Besides, this method contributes to better scalability, since users can increase the computational power with additional processing elements or resources;

- **Resource sharing.**
  There is the need for resource sharing among different parts of distributed system;

- **Scalability.**
  In order to manage with increasing demands on the system, distributed systems could be improved by adding new resources;

- **Fault-tolerance.**
  Powerful uniprocessors may completely collapse in case of the processor fails. In case of distributed systems, it is possible to find a compromise with a partial degradation in system performance in case of a processing elements failure. This is the essence of graceful degradation.

Despite all benefits distributed systems have some disadvantages:

- **Complexity:** generally, distributed systems have more complex structure than centralized systems;

- **Security:** distributed systems are more susceptible to external attacks;

- **Manageability**: due to higher complexity of system management more effort is required;

- **Unpredictability:** distributed system response depends on network load and the system organization that makes it sometimes unpredictable.

Distributed systems are characterized by using the logical or functional distribution of the processing capabilities. The logical distribution is usually based on the following set of criteria [7]:

- **Multiple processes.** The system consists of multiple processes which could be either system or user processes, but each process should have an independent thread of control;

- **Interprocess communication.** Processes communicate with one another using messages that take a finite time to travel from one process to another;

- **Disjoint address spaces.** Processes have disjoint address spaces;

- **Collective goal.** Processes must interact with one another to meet a common goal.

### 2.1.2 Architectures

Due to the large variety of distributed systems characteristics and their general complexity there is no universal architectural model that could serve to all kind of systems. Therefore, there are many types of architects applicable to distributed systems. It is possible to distinguish following major models [25]:

1. Master-slave architecture;
2. Two-tier client–server architecture;
3. Multi-tier client–server architecture;
4. Distributed component architecture;
5. Peer-to-peer architecture.

**Master-slave architectures**

This model is usually applied in real-time systems in which guaranteed interaction response time is required. Besides, needed processing could be easily distributed and localized to slave processors.

The “master” process is responsible for control over “slave” processes. It provides coordination and communication between them. Various “slave” processes perform specific, usually computationally intensive operations.

**Two-tier client–server architectures**

This model commonly used for simple client–server systems and in cases where the system should be centralized for security reasons. In general client-server architecture, the client part of the system runs on user's computer and the server part runs on a remote computer. A two-tier client–server system consists of as a single logical server and a number of clients that use that server.

There are two possible variations of this model:

- In a **thin-client model** client part has only presentation layer whereas all other layers such as data management, application processing, and database are hosted on the server;

- In a **fat-client model** server hosts only database functions and responsible for data management. The other functionalities like application processing and presentation run on clients.

Thus, it could be concluded that the distinction between these two models and the reasons of advantages and disadvantages of each of them is the different balance between distribution of processing power and system management. In a thin-client model, it is simple to manage clients but server could be heavily loaded with processing. In the fat-client model, on the contrary, client's computers processing power is used to distribute some of the workflows. However, in this case, the system requires additional management.

**Multi-tier client–server architectures**

Another type of client-server model which is usually applied in case of high volume of transactions to be processed by the server. In this architecture, problems of two-tier client-server architecture are solved by distribution of presentation, application processing, data management, and database layers to different components of the system. This model allows designing of more scalable systems.

**Distributed component architectures**

This model is usually applied in case when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client–server systems. In this architecture, system is designed as a set of services. Each distributed component implements a service or group of related services. Another part of the system is middleware which manages services and provides communication among them.

The main benefits of distributed component architecture are its flexibility and scalability. The new resources, services, and components could easily be added to the system.

**Peer-to-peer architectures**

This model is commonly applied in case of locally stored information exchange by clients or processing of a large number of independent computations. In decentralized peer-to-peer systems, each connected computer is a node and all nodes are equal components of the system. Some computing tasks require coordination or contact between nodes. In such cases semi-centralized peer-to-peer architecture is used: some of the nodes, “superpeers” take the role of servers.

The main advantage of peer-to-peer architecture is an effective use of resources by distribution of workload among nodes. However, it brings security and trust problem since users open their computers to peer-to-peer communications.

### 2.1.3 Development challenges

Since distributed systems have a very complex structure and a lot of interconnections among distributed resources, they bring a lot of development challenges. Major of them are the following [4, 20, 28]:

1. Object Models and Naming Schemes;
2. Distributed Coordination;
3. Interprocess Communication;
4. Distributed Resources;
5. Fault Tolerance and Security;

**Object Models and Naming Schemes**

This issue needs to be addressed in the beginning of the distributed system design since such important things as a structure of the system, management of the namespace, name

resolution, and access method, are depend on it

**Distributed Coordination**

Processes in distributed need to be coordinated in order to achieve synchronization. There are three major types of synchronization:

- barrier synchronization, which requires each process to reach common synchronization point in order to continue;

- condition coordination, which requires processes to wait for a condition set by other processes;

- mutual exclusion, in which concurrent processes must have mutual exclusion when accessing a critical shared resource.

There are three major problems related to synchronization:

- troubles with obtaining complete state information due to transfer delaying;
- inaccurate or incomplete information due to message transfer delay or failure;
- deadlock of processes, that means that processes are circularly waiting for each other.

**Interprocess Communication**

This is the major problem of distributed systems. The main obstacle is to ensure transparency in communication by finding the way to provide communication methods that could hide physical details of the message passing.

**Distributed Resources**

Mainly, issues related to distributed resources depend on load distribution type: static or

dynamic. In case of static load distribution, also called multiprocessor scheduling, the main concern is minimizing the computation time for the set of related processes. Therefore, the major issue is minimization of communication overhead with efficient scheduling. In case of dynamic load distribution, also called load sharing, the main concern is maximal utilization of set of related processes. Thus, the major issue is process migration. Another important issue related to distributed resources is sharing and replication of data.

**Fault Tolerance and Security**

Distributed systems vulnerability to failures and security threats could be related to faults due to unintentional intrusion or faults due to intentional intrusion. The major issues are trustworthiness of the communicating processes, data and messages confidentiality and integrity, and authentication and authorization.

## 2.2 Quality attributes

A quality attribute measures fulfillment of stakeholders requirements in the system. It is a system property which could be tested and measured. Software architecture design is driven by quality attribute requirements [2, 12].

### 2.2.1 Interoperability

Traditionally, “Interoperability characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other’s services as specified by a common standard” [9].

In order to make functionally compatible components work together, the differences between their implementations should be reconciled. These differences may be related to data or behavioral heterogeneity. In order to provide interoperability, middleware is placed between system's higher and lower levels. However, middleware abstractions define data structure and coordination which is creates requirement to implement various components using the same technology to interoperate.

### 2.2.2 Scalability

The system is scalable if it can be deployed effectively and economically over a range of different "sizes", suitably defined [14].

Scalability can be measured in various dimensions, such as [10, 13]:

1) *Administrative scalability* allows increasing the number of organizations or users, sharing a single distributed system;

2) *Functional scalability* allows the system enhancing by simple adding new functionality;

3) *Geographic scalability* allows maintaining of system efficiency, utility, and usability under the condition of geographic expansion;

4) *Load scalability* allows distributed system to easily expand by modifying, adding, or removing system components, in order to correspond to the changing level of system load.

### 2.2.3 Usability

ISO 9241-11 defines usability as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [26].

Therefore, the following characteristics could be included in the usability definition [15]:

- effective;
- efficient;
- engaging;
- error tolerant.

**Effectiveness**

Effectiveness characterizes completeness and accuracy of achieving user goals. The effectiveness of an interface often relies on the clearly understandable choices presentation. The more informative an interface allows users to better work with it. Effective interface uses appropriate to the task language and close to the user terminology.

**Efficiency**

Efficiency describes the maximum speed in which users can complete their tasks using the system, without accuracy decreasing. Efficiency metrics includes the number of clicks or keystrokes required or the total work time. The large impact on efficiency have navigation design elements: when well-designed, they can significantly decrease the time needed to perform various tasks.

**Engaging**

Engaging interface is enjoyable to use, its major characteristic is proper visual design. It includes the style of the visual presentation, the number, functions and types of graphic images or colors, and the use of any multimedia elements are all part of a user's immediate reaction.

**Error Tolerant**

An error tolerant program characterized by preventing user errors and helping to recover from them in case of occurrence. Error tolerant interface eliminates error-prone conditions or suggests the way to correct the problem.

## 2.3 Software development life cycle of distributed systems

To find the best methodology of developing composites modeling system is necessary to analyze software engineering principles. Starting briefly with software process and then

looking in more details on iterative software engineering approach.

### 2.3.1 Software process

A software process is a set of related activities that leads to the production of a software product [16]. There are various software processes, but all of them includes four fundamental software engineering activities:

1. *Software specification* **-** defined functionality of the software and constraints on its operation;

2. *Software design and implementation* **–** production of the software to meet the specification;

3. *Software validation* **–** ensuring that software satisfies customer requirements;

4. *Software evolution* **–** developing of software according to changing customer needs.

### 2.3.2 Iterative software engineering

With Iterative Development, the project is divided into small parts, which allows developers to obtain users feedback on the earliest stages of development [27]. Commonly, each iteration represents a mini-Waterfall process where feedback from the one phase influence design of the next phase as shown on Fig. 1.

One of the primary characteristics of Iterative Development is its focus on delivering projects in small pieces. Large projects should be divided into group of smaller, individually planned and delivered projects. Thus, each of these projects can be delivered more quickly and in a less structured manner.

The overall flow of the Iterative model is as follows [16]:

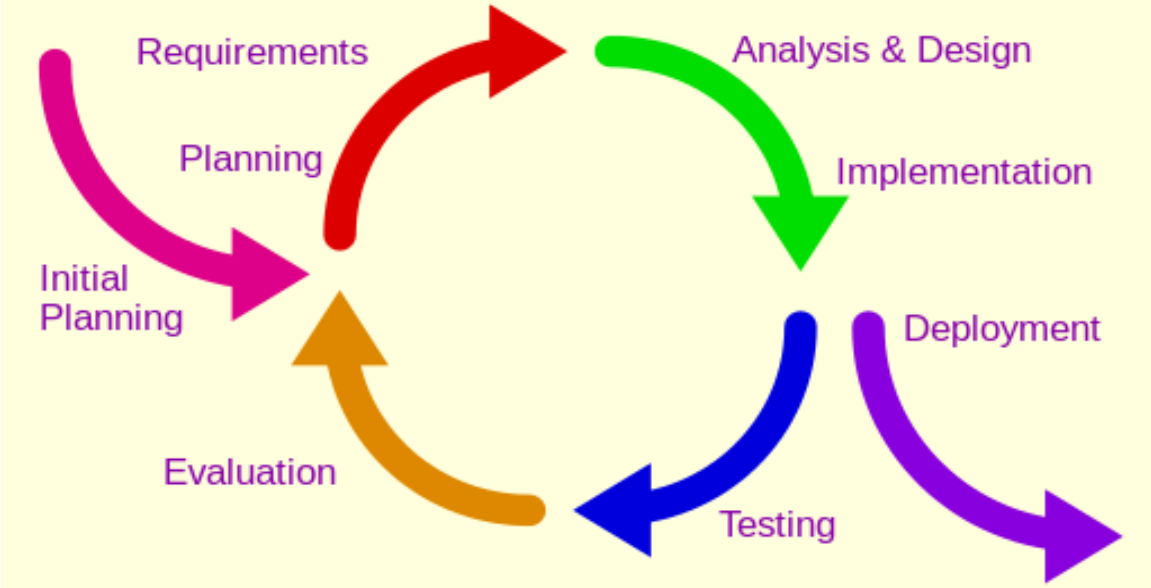


**Fig. 1.** Iterative development model [16]

**1. Planning** consists of describing and defining of the project and managing issues processes;

**2. Analysis:** capturing the business requirements. Firstly they are gathered at a high-level, focusing on the main features and functions to be delivered;

**3. Prototyping:** building application prototype, based on previously received requirements;

**4. Repeating Analysis and Prototyping as Necessary:** completed initial prototype is utilized to gather additional, more detailed business requirements from the client. Once requirements are modified, the prototype is updated to reflect this new set of requirements. After completion of that prototype, it is reviewed by the client to revalidate requirements;

**5. Implementation.**

Iterative Development has many benefits comparing with the traditional waterfall process [18]:

- eliminating of serious misunderstandings early in the life cycle;
- development is focused on most critical project issues and risks;
- iterative testing enables a continuous real-time assessment of the project's status;
- inconsistencies between requirements and implemented system are detected on the earliest stages of the project;
- more even team workload spreading, especially for the testing team;
- enables the team to leverage lessons learned, and, therefore, to continuously improve the process.

However, despite all benefits iterative development is not suitable for all projects. The project is appropriate for this model if it has following main characteristics:

- project size (not too big and not too small);
- manageable design complexity;
- ability to prototype;
- good organization and team flexibility.

# 3 APPLICATION TO COMPOSITE MODELING SYSTEM

This chapter is dedicated to the description of developing process. It reviews the major stages of development which repeat every iteration and presents main results. First of all this chapter more deeply introduces the subject area to the reader. It describes materials modeled by the system, their main properties, and features.

Then, requirement stage is described. This part includes functional and non-functional requirements. After that, analysis of these requirements leads to design of the system. In this chapter in details explained UI prototyping started from wireframing and ending with the final design. Use case diagram illustrates analysis of the system.

Implementation part describes and justifies chosen architecture. And then, gives more detailed specification of classes implemented components of this architecture.

## 3.1 Material description

In case of ceramic materials, there is a graphical model and the problem is importing it from CAD tools to CAE tools for calculations. Due to complex geometry of the model (a large number of curved surfaces), not all surfaces are imported and some of the surfaces are deformed. Method of three-dimensional reconstruction creates finite element mesh without using CAD tools. In this case, the input is the k-file with graphical representation of the model. It contains body border information: coordinates of border dots. It has following structure (Fig. 2, 3).

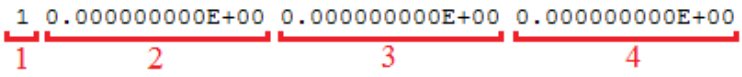


**Fig. 2.** Format of dots description
(1- number of dots, 2- x coordinate, 3 - y coordinate, 4 - z coordinate)

```
*KEYWORD
*NODE
       1 0.000000000E+00 0.000000000E+00-2.000000000E-02       0       0
       2-2.000000000E-02 2.449212708E-18 0.000000000E+00       0       0
       3-4.986138341E-04 0.000000000E+00-1.999378364E-02       0       0
       4-9.969177121E-04 0.000000000E+00-1.997513842E-02       0       0
       5-1.494601870E-03 0.000000000E+00-1.994407594E-02       0       0
       6-1.991356930E-03 0.000000000E+00-1.990061551E-02       0       0
       7-2.486874091E-03 0.000000000E+00-1.984478413E-02       0       0
       8-2.980845321E-03 0.000000000E+00-1.977661652E-02       0       0
       9-3.472963551E-03 0.000000000E+00-1.969615506E-02       0       0
      10-3.962922861E-03 0.000000000E+00-1.960344976E-02       0       0
    .
    .
    .
   21762 2.455371376E-04-1.993953809E-02-1.534448551E-03       0       0
   21763 6.440662082E-04-1.990392173E-02-1.849075079E-03       0       0
   21764 1.535674820E-03-1.991609405E-02-9.954399787E-04       0       0
   21765 1.496676043E-03-1.988460533E-02-1.537020387E-03       0       0
   21766 8.978795296E-04-1.993299765E-02-1.367266279E-03       0       0
   21767-9.284674069E-04-2.101810715E-04-1.997733146E-02       0       0
   21768 4.282482826E-04-5.025709651E-04 1.998909767E-02       0       0
```

**Fig. 3.** Example of input file

The goal of FE reconstruction is to create FE-representation of the model in output k-file (Fig. 4, 5) with the description of FE which composing the body. It consists of coordinates of nodes and FE with their properties. The example of reconstruction is shown on Fig. 6.

```
*KEYWORD
*NODE
       1-2.95906952E+000-1.99000000E+001-2.90000000E+000       0       0
       2-2.85906952E+000-1.99000000E+001-2.90000000E+000       0       0
       3-2.85906952E+000-1.98000000E+001-2.90000000E+000       0       0
       4-2.95906952E+000-1.98000000E+001-2.90000000E+000       0       0
       5-2.95906952E+000-1.99000000E+001-2.80000000E+000       0       0
                                    ...
   79783 2.44093048E+000-1.27000000E+001-2.80000000E+000       0       0
   79784 2.44093048E+000-1.27000000E+001-2.70000000E+000       0       0
   79785 2.44093048E+000-1.26000000E+001-2.90000000E+000       0       0
   79786 2.44093048E+000-1.26000000E+001-2.80000000E+000       0       0
   79787 2.44093048E+000-1.26000000E+001-2.70000000E+000       0       0
*ELEMENT_SOLID
       1       1       1       2       3       4       5       6       7       8
       2       1       5       6       7       8       9      10      11      12
       3       1      13      14      15      16       4       3      17      18
       4       1       4       3      17      18       8       7      19      20
       5       1       8       7      19      20      12      11      21      22
                                    ...
   61371       1   79756   79777   79780   79759   79757   79778   79781   79760
   61372       1   79758   79779   79782   79761   79759   79780   79783   79762
   61373       1   79759   79780   79783   79762   79760   79781   79784   79763
   61374       1   79761   79782   79785   79764   79762   79783   79786   79765
   61375       1   79762   79783   79786   79765   79763   79784   79787   79766
*END
```

**Fig. 4.** Example of output file.


**Fig. 5.** Format of FE description

(1 - number of element, 2 - FE property, 3 - nodes of FE)

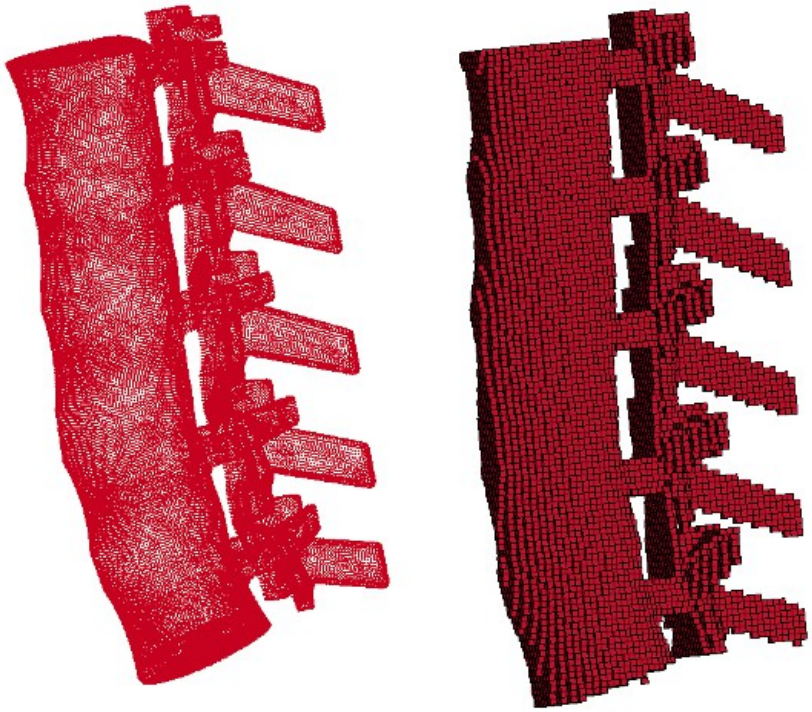

**Fig. 6.** Reconstruction of the spine model

Felt materials or non-woven materials are composed of individual fibers (threads) connected to each other without the use of weaving techniques. The fibers in these materials may be relatively straight or curved and arranged in a random sample volume. Recently becoming popular models where every fiber were simulated individually (with rod or beamed finite elements) since it provides a more consistent and accurate results.

The system developed in this work could create felt material models and glass fiber material models. Examples of calculated models are shown on Fig. 7 and Fig. 9 accordingly.

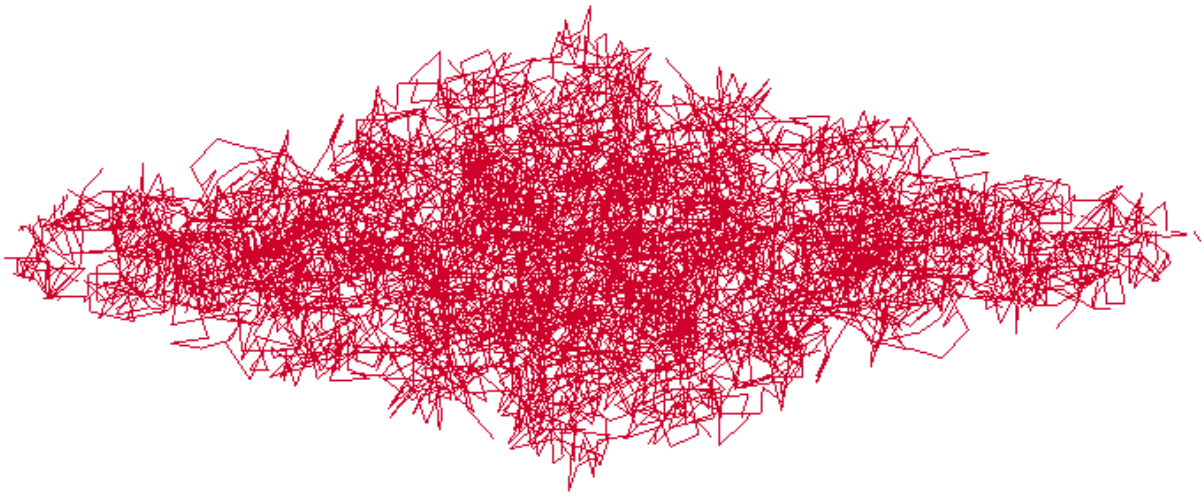

**Fig. 7.** Example of felt material model

Felt materials consist of polynomial threads which have following specific requirements:

- every thread has the same length specified by the user;
- every thread has the same amount of straight parts specified by the user;
- every thread has the same amount of FE specified by the user.

An example of the felt material thread shown on Fig. 8.

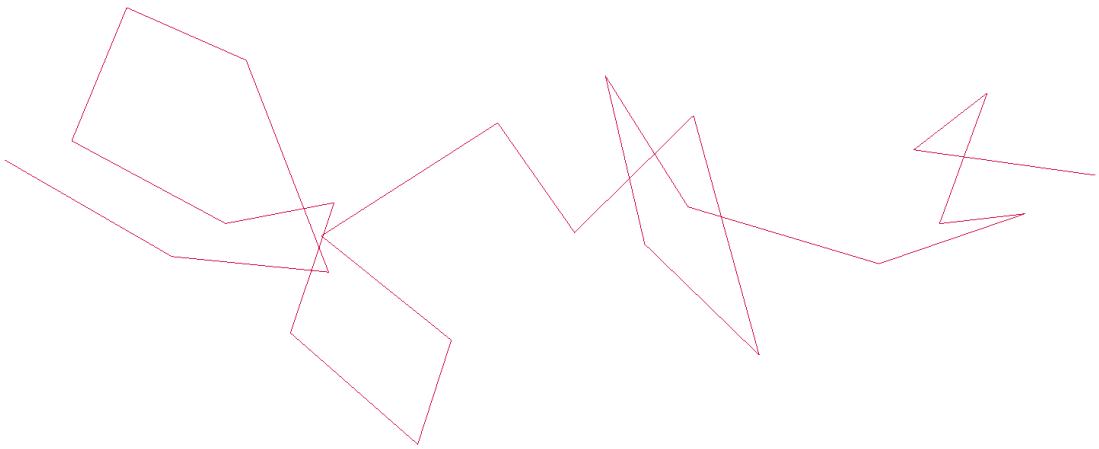

**Fig. 8.** Example of felt material model thread

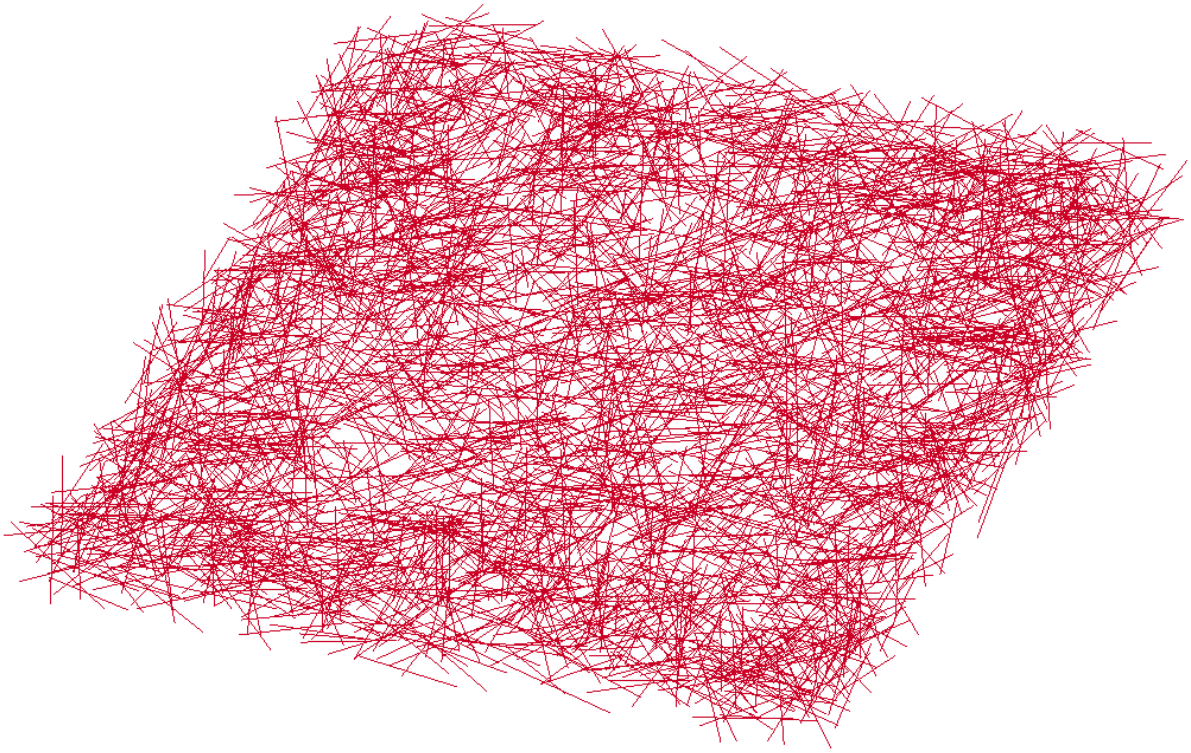

**Fig. 9** Example of glass fiber material model

Glass fiber materials consist of straight threads which have following specific requirements:

- every thread has the same length specified by the user;
- every thread consists of the same amount of finite elements with the size defined by the user.

## 3.2 Requirements

Requirement specification is the most crucial stage in the process of developing software project. Well-defined requirements are needed to ensure the same understanding of the developed system, from the customer and from developers point of view, to clarify the expectations. It serves to understand the planning of the project, what to build and how to validate developed system.

This chapter reviews the set of functional and non-functional requirements gathered through analysis of collected system specifications and requirements demanded by the user.

### 3.2.1 Functional requirements

Requirements which are define functions of the developed system listed in Table 1. For easier reference, each requirement has ID. Priority ranges from low to high, requirements with low priority is possible to neglect in this project.

**Table 1.** Functional requirements

| ID | Requirement | Description | Priority |
|---|---|---|---|
| **FR1** | System should allow user to create (reconstruct) ceramic material model | Finite elements ceramic material model should be reconstructed from the input file with graphical representation of the model | High |
| **FR2** | System should allow user to change material properties of ceramic model | Existing material properties of model's FE could be changed according to ratio given by user | High |
| **FR3** | System should allow user to create felt material model | Felt material model created according to entered model size, length of threads, size of the straight parts, number of FE and number of generated threads | High |
| **FR4** | System should allow user to create glass fiber model | Felt material model created according to entered model size, length of threads, size of FE and number of generated threads | High |
| **FR5** | System should allow user to measure quality of model reconstruction | Quality of model reconstruction should be evaluated from the difference between graphical and FE models | Low |

### 3.2.2 Non-functional requirements

Table 2 presents non-functional requirements which describe overall properties of the system and affect system architecture. Descriptions of requirements will be used to test corresponding qualities.

**Table 2.** Functional requirements

| ID | Requirement | Description | Priority |
|---|---|---|---|
| **NFR1** | Extensibility | System should be able to have new functional components extended. For example, generation of new types of models and new processing functionality | High |
| **NFR2** | Interoperability | System components can co-exist and work together by merely relying on each other's services as specified by a common standard | High |
| **NFR3** | User interface should be simple enough | Minimal amount of clicks needed to achieve goal | High |

## 3.3 User interface prototyping

User interface design was based on the functional requirements analysis. It started from prototyping of wireframes using Balsamiq Mockups which were evolved in the final product.

### 3.3.1 Wireframing

Balsamiq is a rapid wireframing tool which allows users to capture all the skeleton decisions on visual design and user interface implementation [22]. It has a lot of advantages [19, 23]. Most important of them are:

**1.** Easy of use – creation of wireframes right after application opening;

**2.** Simple tools – Balsamiq has intuitive visual tools which help to transfer concepts into design;

**3.** Portability – Balsamiq files could be easily shared across multiple operating systems using e-mail, printing or online.

Besides, Balsamiq allows creating of interactive prototypes which help to test navigation. Fig. 10-12 show interface prototypes created with the help of Balsamiq.

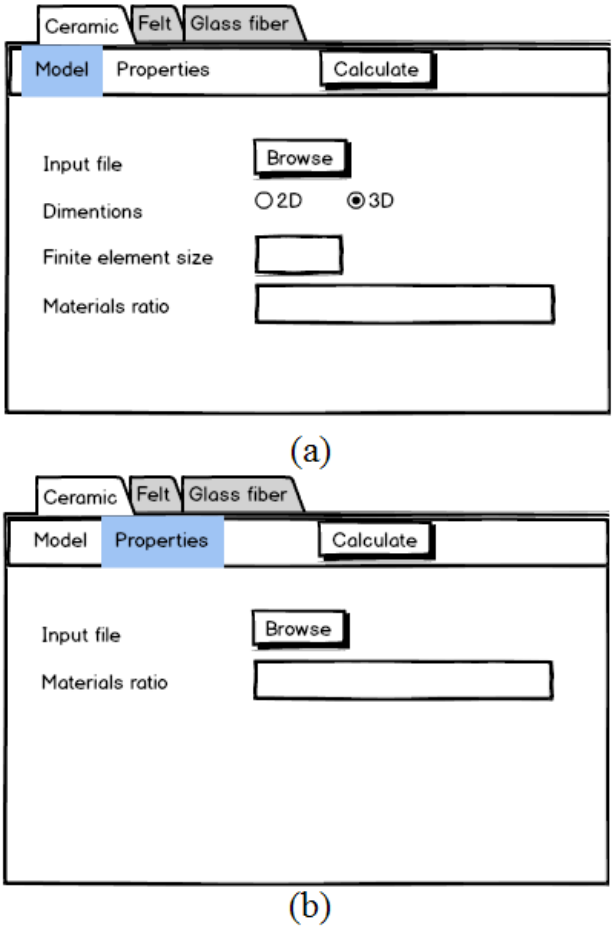


**Fig. 10.** User interface mockups: a) ceramic material models creation; b) properties changing

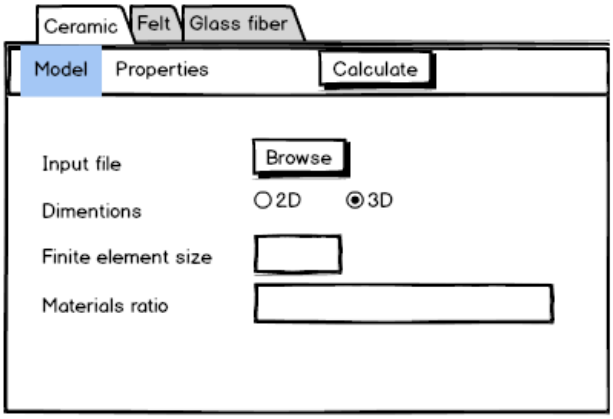


**Fig. 11.** User interface mockups: felt material models

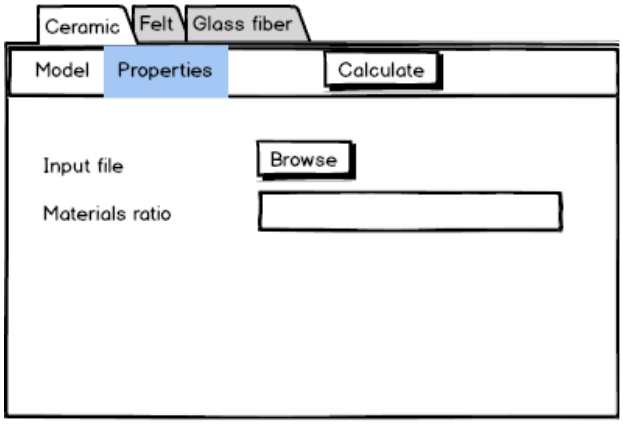


**Fig. 12.** User interface mockups:  glass fiber material models

### 3.3.2 Design

The user interface was designed according to functional requirements. Each of them was analyzed and based on that use case diagram shown on Fig. 13 was created.

There are following main use cases of the system:

1) create ceramic material model – user could reconstruct models from graphical representation;

2) change material properties – user could change material ratio in created ceramic material models;

3) create felt material model;

4) create glass fiber material model.

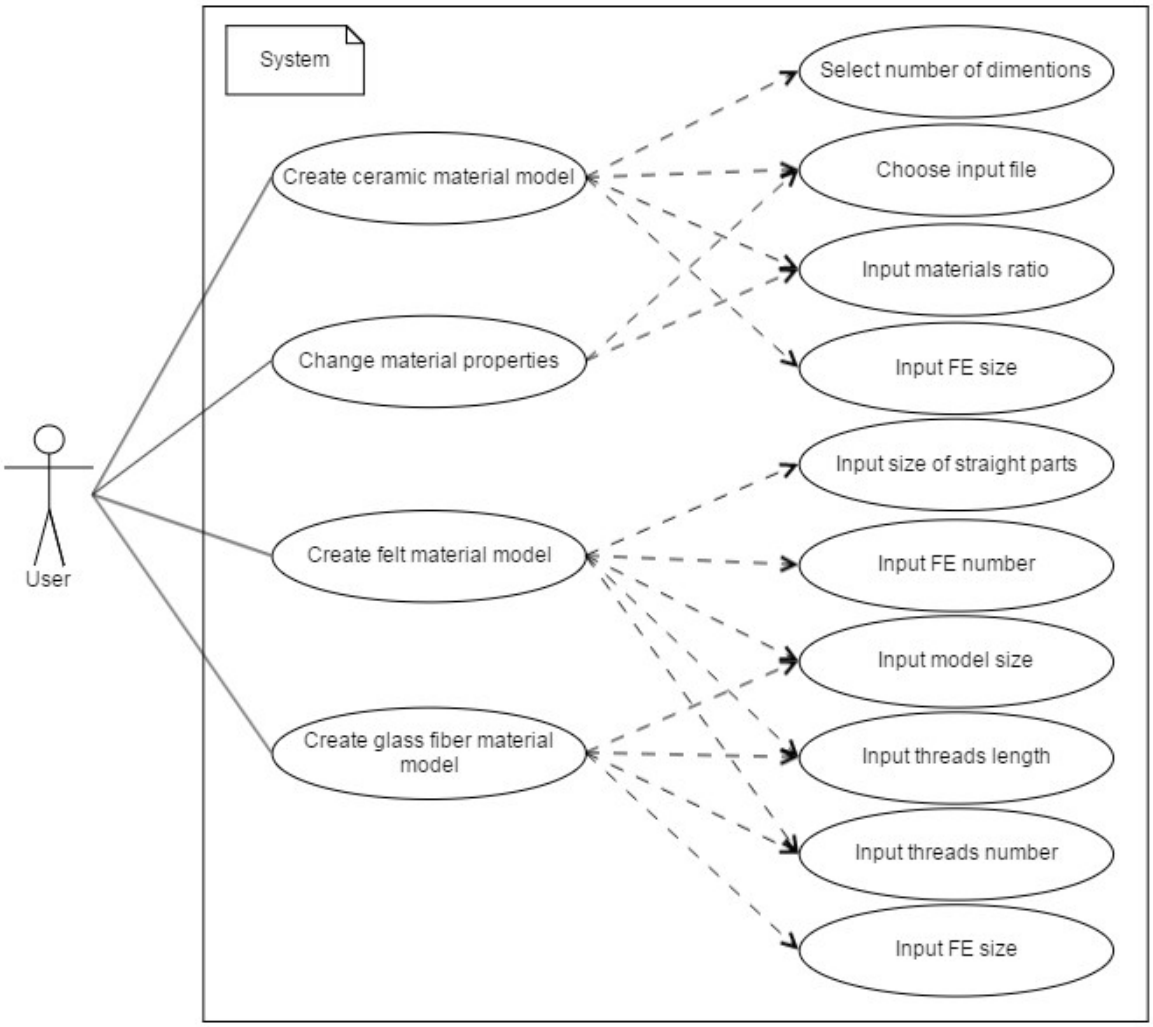


**Fig. 13.** Use case diagram. Dotted line shows relationship <<included>>

Additionally, in this use cases included the ones with defining properties of created models:

- select number of dimensions;
- choose input file;
- input material ratio;

- input finite elements size;
- input finite elements number;
- input straight parts size;
- input model size;
- input threads size;
- input threads number.

**Flow of events**

Single actor of the use case is User of the system who needs to create and process composite models. User chooses the type of model to interact. In order to create a new model user needs to set its parameters such as general properties and FE properties. Based on this input computations are done. Changing material properties starts from choosing ceramic type of models and selecting corresponding functionality. Computation is based on input file and materials ratio.

According to requirements and use case diagram, based on the prototypes, following user interface was created [Fig. 14-16].

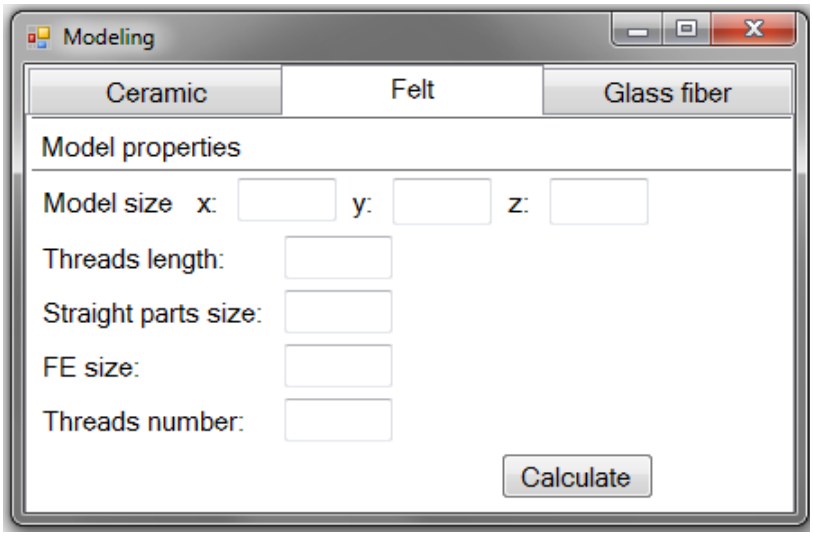


**Fig. 14.** User interface for felt material models creation

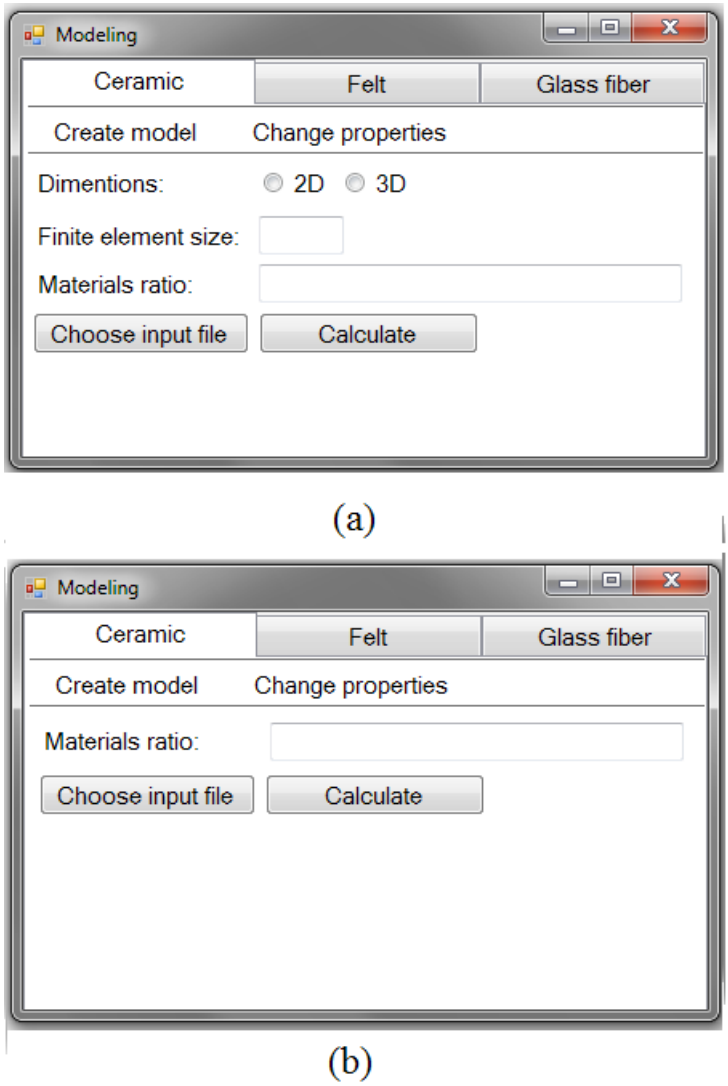


**Fig. 15.** User interface for ceramic material models: a) creation, b) properties changing

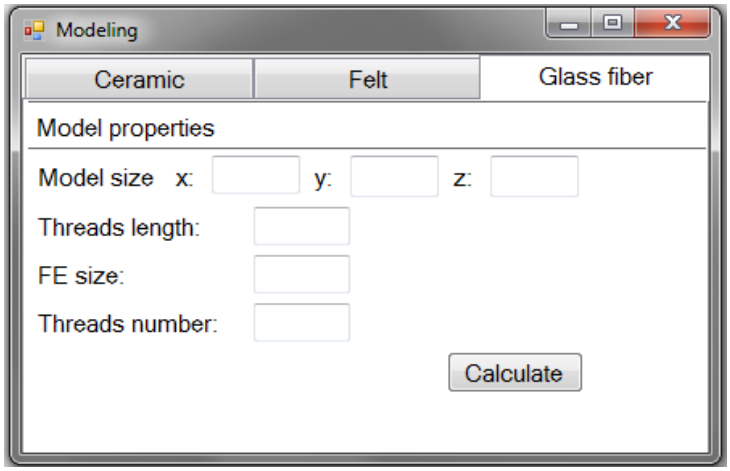


**Fig. 16.** User interface for glass fiber material models creation

## 3.4 Implementation

The system was implemented according to developed architecture. The basic architectural view is shown on Fig. 17. For the studied system was chosen service-oriented architecture which allows developers to increase flexibility and integrate heterogeneous components [24].

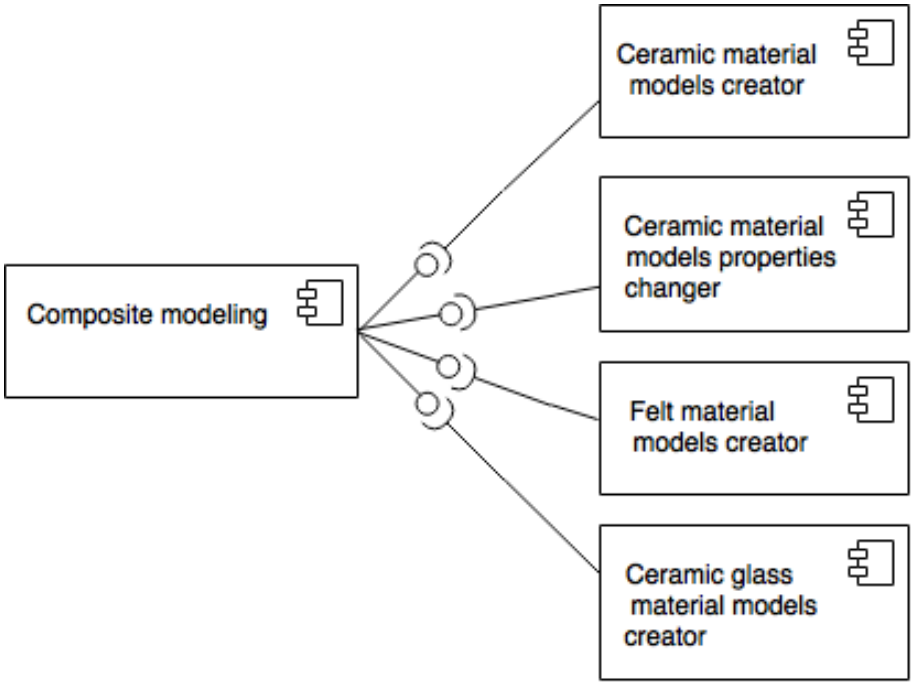


**Fig. 17.** Basic architecture

SOA has many benefits which make it very suitable for developed system based on its requirements. The major advantages include:

- Interoperability of system components and, besides, interoperability with legacy application;
- Enabling just-in-time integration;
- Reducing complexity by encapsulation.

The system consists of five main components:

1) **Composite modeling** presents the shell of the system: it provides user interface;

**2) Ceramic material models creator** consists of legacy classes providing functionality related to ceramic models reconstruction;

**3) Ceramic material models properties changer** consists of classes providing functionality related to changing material properties of ceramic models;

**4) Felt material models creator** consists of classes providing functionality related to felt models creation;

**5) Fiber glass material models creator** consists of classes providing functionality related to glass fiber models creation;

Components communicate between each other through messages.

# 4 TESTING AND EVALUATION

This chapter describes conducted tests and usability evaluation which were performed in order to assure that developed system satisfies specified requirements.

## 4.1 Testing

In order to eliminate errors and evaluate quality of the developed system in each iteration set of tests were conducted. This set includes three types of tests for three levels of system abstraction:

**1.** Unit tests on modules level to verify working ability of individual units of functionality, beginning from the smallest, like methods and finishing with the set of classes;

**2.** Integration testing which assures that combination of modules verified by unit tests working correctly;

**3.** Functional testing on the highest level done as requirements-based testing to validate implementation of all functionality required by the user. It was done by executing each use case, using valid and invalid data, to verify the following:

- the expected results occur when valid data is used;
- the appropriate error or warning messages are displayed when invalid data is used;
- each business rule is properly applied.

Test cases presented in tables 3-6.

**Table 3.** TC1

| **ID** | TC1 | |
|---|---|---|
| **Function** | Creation of ceramic material models [FR1] | |
| **Action** | **Expected result** | **Test result (passed/failed/blocked)** |
| **PreConditions** | | |
| Launch modeling application | Main interface page is opened and available | Passed |
| **Test case description** | | |
| Open Ceramic materials tab | Ceramic materials interface is available | Passed |
| Enter non-numerical FE size | Warning message “inappropriate FE size” | Passed |
| Enter wrong materials ratio (total not equal to 100% ) | Warning message “wrong materials ratio” | Passed |
| Enter valid attributes and calculate model | Message “Model successfully generated” | Passed |
| Open view of the model | Graphical view of the model is opened | Passed |

**Table 4.** TC2

| **ID** | TC2 | |
|---|---|---|
| **Function** | Changing material properties [FR2] | |
| **Action** | **Expected result** | **Test result (passed/failed/blocked)** |
| **PreConditions** | | |
| Launch modeling application | Main interface page is opened and available | Passed |
| **Test case description** | | |
| Open Ceramic materials tab | Ceramic materials interface is available | Passed |
| Enter wrong materials ratio (total not equal to 100% ) | Warning message “wrong materials ratio” | Passed |

**Table 4** (continues)

| Action | Expected result | Test result (passed/failed/blocked) |
|---|---|---|
| Enter valid attributes and calculate properties | Message “Properties changed” | Passed |
| Open view of the changed model | Graphical view of the model is opened | Passed |

**Table 5.** TC3

| **ID** | TC3 | |
|---|---|---|
| **Function** | Creation of felt material models [FR3] | |
| **Action** | **Expected result** | **Test result (passed/failed/blocked)** |
| **PreConditions** | | |
| Launch modeling application | Main interface page is opened and available | Passed |
| **Test case description** | | |
| Open Felt materials tab | Felt materials interface is available | Passed |
| Enter non-numerical model size | Warning message “inappropriate model size” | Passed |
| Enter numerical thread length | Warning message “inappropriate thread length” | Passed |
| Enter thread length bigger than model size | Warning message “thread length is too big” | Passed |
| Enter non-numerical straight parts length | Warning message “inappropriate straight parts length” | Passed |
| Enter thread straight parts length bigger than thread length | Warning message “straight parts length is too big” | Passed |

**Table 5** (continues)

| Action | Expected result | Test result (passed/failed/blocked) |
|---|---|---|
| Enter non-numerical FE size | Warning message "inappropriate FE size" | Passed |
| Enter FE size bigger than straight parts length | Warning message "FE size is too big" | Passed |
| Enter non-thread number FE size | Warning message "inappropriate thread number" | Passed |
| Enter valid attributes and calculate model | Message "Model successfully generated" | Passed |
| Open view of the model | Graphical view of the model is opened | Passed |

**Table 6.** TC4

| **ID** | TC4 | |
|---|---|---|
| **Function** | Creation of glass fiber material models [FR4] | |
| **Action** | **Expected result** | **Test result (passed/failed/blocked)** |
| **PreConditions** | | |
| Launch modeling application | Main interface page is opened and available | Passed |
| **Test case description** | | |
| Open Glass fiber materials tab | Glass fiber materials interface is available | Passed |
| Enter non-numerical model size | Warning message "inappropriate model size" | Passed |
| Enter non-numerical thread length | Warning message "inappropriate thread length" | Passed |
| Enter thread length bigger than model size | Warning message "thread length is too big" | Passed |

**Table 6** (continues)

| Action | Expected result | Test result (passed/failed/blocked) |
|---|---|---|
| Enter non-numerical FE size | Warning message "inappropriate FE size" | Passed |
| Enter FE size bigger than thread length length | Warning message "FE size is too big" | Passed |
| Enter non-thread number FE size | Warning message "inappropriate thread number" | Passed |
| Enter valid attributes and calculate model | Message "Model successfully generated" | Passed |
| Open view of the model | Graphical view of the model is opened | Passed |

### 4.2 Heuristic usability evaluation

For the purpose of interface evaluation the Nielsen's Heuristics [21] was used. It includes evaluation of 10 principles of user interface:

- **[N1] Visibility of system status:** the system should always give users appropriate feedback within reasonable time;
- **[N2] Match between system and the real world:** the system should use natural language, understandable to the user;
- **[N3] User control and freedom;**
- **[N4] Consistency and standards:** user interface should be consistent in terms of used terminology and presentation;
- **[N5] Error prevention:** design of interface should prevent a problem from

occurring in the first place.;

- **[N6] Recognition rather than recall:** instructions for use of the system should be visibleUniqueeliminating of serious or easily retrievable;

- **[N7] Flexibility and efficiency of use:** accelerators may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users;

- **[N8] Aesthetic and minimalist design:** dialogs should not contain irrelevant information;

- **[N9] Help users recognize, diagnose, and recover from errors:** error messages should indicate the problem and suggest a solution, using natural language;

- **[N10] Help and documentation.**

For the heuristic evaluation were used rating scale from 0 to 3, where 0 is missing, 1 is poor, 2 is fair and 3 is good. Its results are shown in Table 7. The average of all final heuristic evaluations is 1,8, from which could be concluded that the usability quality is fair. The weakest points related to flexibility, control and user help.

**Table 7.** Results of usability evaluation

| ID | Heuristic | 1st expert | 2nd expert | 3rd expert | Final evaluation |
|---|---|---|---|---|---|
| N1 | Visibility of system status | 3 | 3 | 3 | 3 (good) |
| N2 | Match between system and the real world | 3 | 2 | 2 | 2,3 (fair) |
| N3 | User control and freedom | 1 | 1 | 1 | 1 (poor) |
| N4 | Consistency and standards | 3 | 3 | 3 | 3 (good) |
| N5 | Error prevention | 2 | 3 | 2 | 2,3 (fair) |
| N6 | Recognition rather than recall | 2 | 1 | 2 | 1,7 (fair) |

**Table 7** (continues)

| ID | Heuristic | 1st expert | 2nd expert | 3rd expert | Final evaluation |
|---|---|---|---|---|---|
| N7 | Flexibility and efficiency of use | 0 | 0 | 0 | 0 (missing) |
| N8 | Aesthetic and minimalist design | 3 | 2 | 3 | 2,7 (good) |
| N9 | Help users recognize, diagnose, and recover from errors | 2 | 2 | 1 | 1,7 fair) |
| N10 | Help and documentation | 0 | 0 | 0 | 0 (missing) |

# 5 RESULTS AND RECOMMENDATIONS

The used software engineering method takes into account the major features of distributed systems, their main disadvantages such as complexity, security, manageability, unpredictability and development challenges related to it. Chapter 2.3 studies the principles of software engineering, stages of software life cycle and driven from distributed systems characteristics, chosen iterative development process. Gathered knowledge were further applied during developing of the modeling system in chapter 3. This chapter also covers the topic of composite modeling and its features. Implemented system was tested and evaluated in chapter 4 which provides assurance that developed system satisfies declared requirements.

# REFERENCES


1. Balsamiq [Online].
URL: https://balsamiq.com/ (Accessed: 18 July 2015).

2. Bass, L., Clements, P., Kazman, R. 2012. Software Architecture in Practice. Third Edition, Addison-Wesley, pp. 63.

3. Buschmann, F., Henney, K., Schmidt, D.C. 2007. Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing. Wiley and Sons, pp. 17-24.

4. Chow, R., Johnson, T. 1997. Distributed Operating Systems & Algorithms. Addison-Wesley, pp. 47-50.

5. Coulouris, G., Dollimore, J., Kindberg, T., Blair. G. 2011. Distributed Systems: Concepts and Design (5th Edition). Boston: Addison-Wesley, pp.14-15.

6. Dolev, S., 2000. Self-stabilization, MIT Press, pp. 5-7.

7. Dr. Trkman, P., Dr. Kovačič, A., Dr. Popovič, A., SOA Adoption Phases, *Business & Information Systems Engineering,* Volume 3, Issue 4, August 2011, pp 211-220.

8. Faranello, S. 2012. Balsamiq Wireframes Quickstart Guide, Packt Publishing.

9. Garrett, J.J. 2011. The Elements of User Experience: User-Centered Design for the Web and Beyond, Sagebrush Education Resources, pp. 128-129.

10. Gorthon, I., 2011. Essential Software Architecture, Springer-Verlag Berlin Heidelberg, pp. 27-30.

11. Ha-Minh, C., Imad, A., Kanit, T., Boussu F. Numerical analysis of a ballistic impact on textile fabric, *International Journal of Mechanical Sciences*, vol. 69, 2013, pp. 32-39.

12. Ha-Minh, C., Kanit T., Boussu F., Imad T., Numerical multi-scale modeling for textile woven fabric against ballistic impact, *Computational Materials Science*, vol. 50, 2011, pp. 2172–2184.

13. Hevner, A., Chatterjee, S. 2010. Design Research in Information Systems, Springer US, pp. 12.

14. ISO 9241-11:1998, Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability.

15. Issarny, V., Bennaceur, A., Composing Distributed Systems: Overcoming the Interoperability Challenge, Inria Paris-Rocquencourt, France, pp. 2-3.

16. Iterative Development Lifecycle [Online].
URL: http://www.lifecyclestep.com/browse/462.0RADLifecycle.htm (Accessed: 12 June 2015).

17. Jogalekar, P., Woodside, M., Evaluating the Scalability of Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, 2000, pp. 1-2.

18. Kruchten, P., From Waterfall to Iterative Development – A Challenging Transition for Project Managers, *The Rational Edge*, 2001.

19. Larman, C., Basili, V.R., Iterative and Incremental Development: A Brief History, *IEEE Computer Society,* Volume 36, Issue 6, June 2003, pp 47-56.

20. Nesset, D., Massively Distributed Systems: Design Issues and Challenges, *Proceedings of the Embedded Systems Workshop*. March 1999, pp. 5-15.

21. Nielsen, J., 10 Usability Heuristics for User Interface Design [Online].
URL: http://www.nngroup.com/articles/ten-usability-heuristics/ (Accessed: 5 November 2015).

22. Peffers, K., Tuunanen, T., Gengler, C.E., Rossi, M., Hui, W., Virtanen, V., Bragge, J., The design science research process: a model for producing and presenting information systems research, *Proceedings of the first international conference on design science research in information systems and technology*, pp. 83-106, 2006.

23. Quesenbery, W. 2011. What Does Usability Mean: Looking Beyond 'Ease of Use'. 48th Annual Conference, Society for Technical Communication.

24. Software Engineering Institute. Software Architecture Technology Initiative [Online].
URL: http://www.sei.cmu.edu/architecture/sat_init.html (Accessed: 11 June 2015).

25. Sommerville, I. 2011. Software engineering, Addison-Wesley, pp. 27-29.

26. Sukumar, G., 2007. Distribute Systems. An Algorithmic Approach, Taylor & Francis Group, pp. 3-5.

27. Tannenbaum, A.S., Van Steen, M. 2006. Distributed systems, Pearson, pp. 9.

28. Tcui, T., Dr. Jeng, A.B. 2009. Challenges and Solutions of Distributed Systems Composition. Telecom Technology Center.