TEACHING TO WRITE SOFTWARE ENGINEERING DOCUMENTS WITH FOCUS ON DOCUMENT DESIGN BY MEANS OF MIND MAPS

Dmitrij V. Koznov
Saint-Petersburg State University, Mathematics and Mechanics Faculty
28 Unversitesky pr., Saint-Petersburg, 198504, Russia
dim@math.spbu.ru

ABSTRACT
Students in our Software Engineering Bachelor's/Master's program had good programming skills, however, wrote technical documents poorly. In order to solve this problem, I have developed a technique to teach writing software engineering documents, focusing on documentation design with mind maps. I used this technique in Introduction to Software Engineering course (SE201) and found out that it will considerably improve quality of student technical writing skills.

KEY WORDS
Teaching software engineering, software engineering documentation, design of documentation, mind maps

1. Introduction

In most computer science courses taken prior to Software Engineering, professor provides detailed requirements and design to keep students concentrated on issues of implementation that may appear extremely complex to them. After a couple of years of practice with this model, students become proficient at implementation of instructor's tasks and wrongly think that software implementation is coding entirety. Unless this wrong belief is timely dispelled, then in the future it is very hard to convince students of necessity requirement analysis, architecture design, and other software engineering activities.

Development of documentation (both internal and external) is an important activity in the software project. But students learn to write technical documents reluctantly, and in many software engineering courses this issue is omitted, or implemented in a very straightforward manner: professor provides students with templates and they have to fill them out only.

At development of technical documentation a lot of attention is paid to design of document's structure [1, 2]. However, documentation design approaches lacked in software engineering courses at the moment. Meanwhile, such approaches could simplify teaching to write documents since a lot of mistakes can be detected and corrected before the writing itself, and it will take less effort to give feedback and make corrections: reading and commenting on document design is much easier than texts; changing design is faster than rewriting the text. Moreover, designing documents, it is possible to make process of learning less boring and more attractive.

In the given paper document design technique for teaching to write software engineering documents is presented. The technique is based on mind maps [3] and online mind map tool Comapping [4]. The technique was used in Software Engineering course (SE201) and provided considerable improvement of student technical writing skills.

2. Background

2.1 Approaches for teaching to write software engineering documents

University courses on software engineering could be divided roughly into following classes: initial programming courses [5], introductory software engineering courses [6, 7], and senior courses [8]. Initial programming courses focus on programming skills only. Introductory courses consider software engineering as a whole and are usually assisted by different assignments and samples, and often are project-oriented. The latter implies that students work in small groups around software projects and perform of different activities of development lifecycle. There is a risk organizing introductory courses in project-oriented manner, since some group members with less coding confidence would devote more time to documenting and other noncoding tasks, while students that enjoy coding would carry out a major part of implementation [9]. Therefore, it's hard to ensure that all of the students are taught software engineering skills evenly. In this case, group rotation method can help [10], however, I think it is better to familiarize students with initial software engineering skills using lectures and assignments, followed by integration of knowledge received into student projects in further courses.

Documentation writing is usually included in both introduction and senior software engineering courses. A lot of attention is paid to development of different kinds of software documentation using samples and templates [7, 8]. Let's analyze, how writing software engineering documentation is taught in software engineering courses.

In [11] approach to teaching software engineering using open source software is presented. During class

students create requirement specifications, design documentation, and different reports. In addition, students employ open source software and are able to receive actual estimation of the documentation importance, as such documents are critically vital in understanding software.

In [12] the teaching program on software engineering in Kaunas University of Technology is overviewed. At that, written documents and oral presentations are one of the main objectives of the program: students write test specifications and various project reports. Teaching of software engineering is organized on the base of project-oriented method.

An approach to teaching software development skills early in the curriculum is presented in [6]. The approach focuses on teaching software development skills that are valuable in other upper level computer science and computer engineering courses. The course presented is project-oriented. During the project students develop following documentation: requirement specification, architecture design, and test plan.

Paper [13] reports an experience of incorporating XP[1] into an existing document-centric software engineering course. Firstly, initial course COMP389 is described. After that XP correction of the course is presented. COMP389 is project-oriented. Every 2-3 weeks (of a 12 week course) each student group complete a document to meet a deadline, beginning with a project plan and finishing with deliverable software to external clients. These clients were either academic or general staff, often without any particular knowledge of software development. Among other things, clients are invited to comment on all documents produced by the groups. Each project group is assigned of a tutor. One of the tutor's task is to perform proofreading, mentoring, and assessing teams on creation of the documents they must produce — each document is handed in twice: a draft one week prior to the deadline, and then a revised version, which contributes to the final grade. Totally 80% of the project grade is for document production. So much effort to create the documentation made the course very time-consuming and rather boring for students, as noted by the authors themselves. Introducing XP features and reducing documentation work made the course interesting and more appropriate to modern software engineering tends.

In [10] group rotation method for student software engineering projects is presented. Students form different teams at each milestone. In particular, one of the approach benefits is that project documents of each milestone are more complete and are of much better quality. Group rotation method was used by different authors (see, e.g. [15, 16]).

A writing class technique to create software design documents in the context of software engineering course is presented in [8]. There are following innovations of the technique:

- Require the students to turn in drafts, which have two grades: an actual grade for the draft and a hypothetical grade assuming what the final product would have been, basing on this draft. In this case students won't be surprised by a poor grade on the final project after getting a good grade on the draft.
- Use four or more different samples/templates; otherwise, students often focus on idiosyncratic features from one sample or template and introduce that into their product.

Paper [9] reports about project-oriented senior two-term software engineering course aiming to overcome on-the-job training problem. The course is addressed to the following topics: team-based development, problem definition from real-world clients, managing development process, and others. The main tasks in the first term are mostly documentation-related: each group generates use cases, scenarios, nonfunctional requirements, acceptance plan, a schedule, a release plan, and other documents. The result of the first term is detailed analysis and design of the final product for the second term. Documentation writing skills were one of the most significant course outcomes, which students highlighted in their feedback.

In [7] three course computer science introductory sequence is presented. The main goals of the sequence are to introduce software engineering concepts early, provide a consistent software engineering focus from one course to the next, and prepare students to senior project-oriented courses. Introductory sequence incorporates the following concepts: the need for software engineering; characteristics of good software, software development process, specification of requirements, software design quality and documentation, testing plans and techniques; maintenance; team work. The courses are a mixture of lectures and assignments. The former include document writing, e.g.: (i) present students with a design documentation and code, written by others, and assign them to structurally test that code, (ii) provide students with requirements for a system and ask them to produce a high-level design without giving them details of physical implementation of the system, (iii) use design documents written by others to teach students to assess, compare, and critique designs, (iv) allow students to work in small teams to develop various portions of the systems and related documentation.

It should be noted that teaching to write software engineering documents uses different kinds of reviews, assessments, and discussions. A lot of attention is also

[1] Extreme programming (XP) is one of the most famous agile software development method, that is intended to improve software quality and responsiveness to changing customer requirements. Extreme programming has been described as a set of practices: pair programming, planning game, test-driven development, continuous integration, refactoring or design improvement, small releases, coding standards, collective code ownership, simple design, system metaphor, the customer is always available, no overtime [14].

paid to samples and templates, but it lacks documentation design techniques.

2.2 Mind Maps

The approach was suggested by Tony Buzan in 1970 as an efficient way to work with arbitrary information [3]. The idea is to use a very simple diagramming notation: the central (primary) object is drawn in the middle, and secondary and further objects, which clarify the meaning of the central one, are put around it and connected together (see Fig. 1).

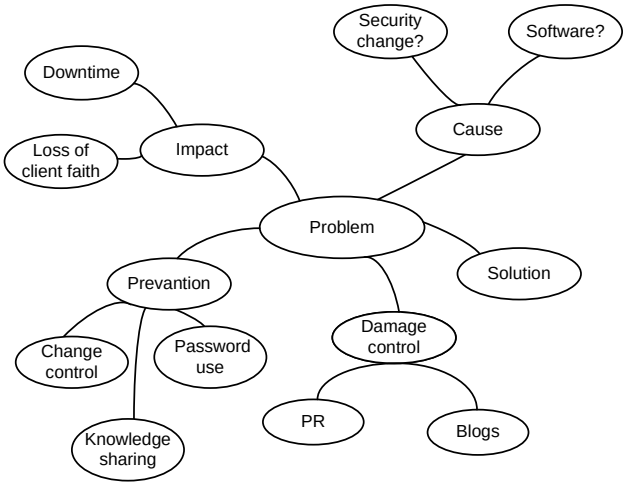

Figure 1. An example of mid map

This makes a radial structure suitable for analyzing and understanding large amounts of data. The approach is widely used in education, business, psychology, and other areas. There are also a number of software tools implementing the approach. More information can be found in [3].

However, despite simplicity of the method and notation, successful use of mind maps in education requires additional effort. One has to determine what kinds of information are suitable to depict with mind maps, and what further effect ones will have on students.

2.3 Comapping

Comapping [4] is a collaborative online mindmapping application. Its basic purpose is to allow users to create, edit, and share mind maps. Comapping introduces features like easy drag and drop, smooth animation, support of large maps with smart auto-focusing features, and more. Tree-like notation (left-to-right mindmapping) is better for computer-based support when combined with layout algorithm, as it is easier to read and understand than the center-based one. An example of a mind map in Comapping is shown in Fig. 2.

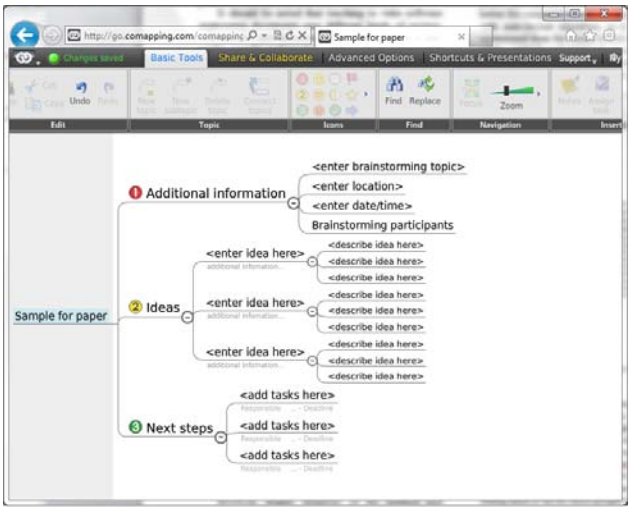

Figure 2. Example of a mind map diagram in Comapping

The system allows sharing maps to any number of users, as well as notifying users about map changes. These changes are then highlighted on a map with detailed information on who and when made changes. Other users can review the changes and leave comments, making Comapping a good tool for professors to review student works.

3. SE201 overview

The course was aimed to introduce Software Engineering as a whole, familiarizing students with software development process and different kinds of activities, outlining various professions (developer, tester, manager, etc.) and opportunities for IT-career development.

The course also covered requirement engineering, configuration management, and testing. It also had emphasis on development of communication skills: a number of assignments were performed in groups, students made presentations, and talked extensively with me and the course assistant. The course lasted for 17-19 weeks, 4 hours per week and was held four times (2008-2011). Each year it was attended by 15-20 people. This course was preceded by Introduction to Programming (CS101) course, so the students have already had initial programming skills.

Documentation training was one of the main practical assignments of the course, which was launched due to the following reasons:

- Documentation development skills are essential for effective completion of university education (term and course papers, thesis, etc.).
- Traditionally, at our faculty there is contempt for paper work; idea and solution of the problem are

considered important, students studying a large number of mathematical disciplines in their first years, are imbued with these views.

- At teaching documentation development skills we succeed to naturally introduce students to a number of basic software engineering topics: requirements, architecture, usability, maintenance and support, etc.

4. Documentation training

4.1 Overview

General scheme of documentation training is presented in Fig. 3. Training starts with a lecture on software documentation. Students are explained that this kind of activity cannot be entirely handed over to a technical writer, and that programmer, manager, tester, and other project participants should be able to create easily and fast software documents.

Different kinds of software documents are observed, documentation lifecycle and activity kinds, contemporary concepts and development tools are considered. Specifics of different document types are presented, as well as the importance of a good document structure are discussed.

At the next phase (Preparation), software is discussed, that students will describe in their documents. Students also explore mind maps and Comapping. Learning infrastructure of the course is established.

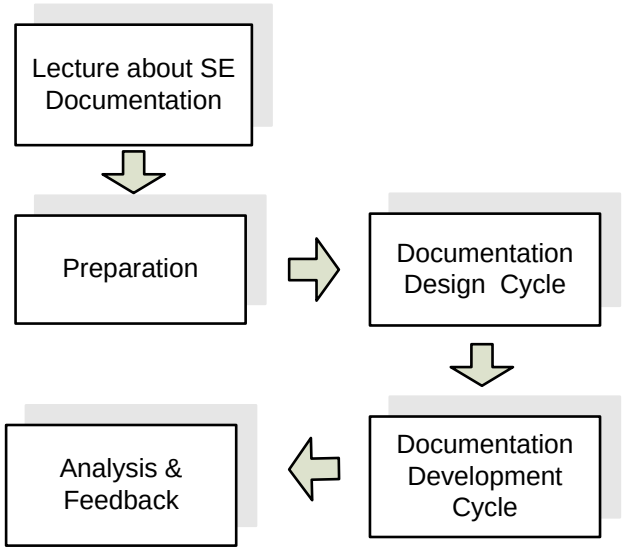

Figure 3. Schema of documentation training

After that, students design documents using mind maps/Comapping (Documentation Design Cycle). Students make several iterations, I and my assistant review and comment results.

Next, students write documents, using the developed design (Documentation Development Cycle). Several iterations are also conducted.

Finally, I make a summary and students provide feedback (Analysis & Feedback).

4.2 Preparation

The scheme of this phase is presented in Fig. 4. Previous experience of this course has shown that in order to make the efficient learning process, students should create documentation for software they have developed, as it will be in the future when they are working at a software industry. I suggested that students create documentation for applications that were developed in Introduction to Programming course.

After that, students are explained of the documents specifics they need to create (Explanation of Document Specifics). Students are suggested to develop three documents: requirement and architecture specifications, a user manual. It is important that they create three documents, rather than one, as each document has its specifics, and students understand better each type of the document producing better texts. Following [8], I gave more than four samples for each type of document, otherwise, the students try to apply the patterns shown as is, even if they do not fit (in whole or in part) to their situation.

Further I identify the students' knowledge of requirements, software architecture and UML, software maintenance and support[2]. Despite the fact that these issues are considered in detail later in the curriculum, they are also discussed in this course, first, because without it students will not be able to create documents properly, and second, repeated return to the same subject in different courses improves absorption of the information. Based on students' initial knowledge and skills they have, I conduct a number of additional assignments. In particular, in 2011, as part of the course, set of assignments on the functional requirements were conducted.

Next, students master the mind maps and Comapping (Mind Maps/Comapping Teaching). These skills are then used in designing documents.

Development Learning Infrastructure is explanation of procedures for interaction between a professor and students throughout the course. For example, regular homework assignments are essential part of the course. Since the group of people is quite large (up to 20 people), it is necessary to follow strict rules in order to work effectively: exactly one day before class, students

[2] Each year, the student groups differ from one another, as the curriculum is constantly being upgraded. In addition, students receive a lot of additional knowledge in practical classes, where much depends on the teacher, therefore, various side knowledge, that different teachers equip their courses, is not strictly regulated.

must send an e-mail with completed assignments, I print them, read, and then give comments during class.

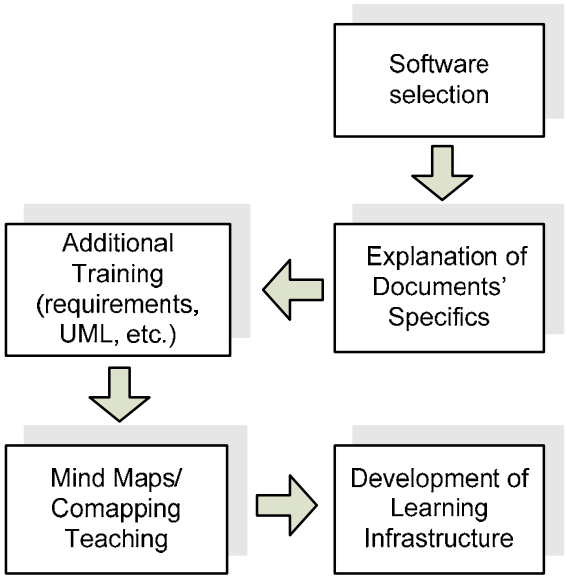

Figure 4. Preparation phases of the documentation training

If students send their works discordantly, it would be difficult to find time to review them. Experience shows that students do not learn this rule at the first push.

4.3 Document Design Cycle

In the early years of the course, requirement specifications, architecture specifications and user manuals, created by the students were almost identical to each other and consisted mainly of fragments of the initial software specifications, handed out by the professor of Introduction to Programming course. In order to solve this problem, I offer each student to create three types of documents (previously, each created only one document). I have also included Document Design Cycle in the training, which consists of three steps: Definition of Documents' Focus, Collection and Elicitation of Information, Development of Document Plan. Final results of Document Design Cycle were presented as mind maps, split into three branches. As software systems, for which the documents were created, were small, then even mind maps in expanded mode were almost entirely displayed on the one screen of computer monitor. It was very useful for students to see the difference between three dissimilar documents on a one view.

The first step is Definition of Documents' Focus: understanding how documents will "work" and modeling of readers.

Each of the proposed documents should "work" in various ways: requirement specification is intended to specify features, which should be implemented, and conform them between developers and the customer. Architecture specification that in our course is a mixture of design specification and internal code documentation addresses for programmers who will carry out maintenance and support. User manual is intended for software users and should comprehensively explain how the software must be used.

To determine the documentation readers, personas design of Alan Cooper [17] was used. It is important that students clearly understand the readers of their documents (and beyond — the actual users of their software!). Customers, programmers, and users of the system are absolutely different persons related to software and development process. They have different background in IT, diverse business, and viewpoint at the system. Students tried to imagine project stakeholders and model a typical reader for each document, following the recommendations of Alan Cooper, and my comments.

At the second step, basing on the identified above focuses, students gathered information for the documents eliciting functional and nonfunctional requirements, recovering architecture, determining software sceneries and user interface features. This was done using mind maps and Comapping, which proved quite suitable for that purpose, as the information has a discrete nature and hierarchical structure. Each student made 2–3 iterations.

Focus detection and information collection steps allowed students to effectively perform document design. Otherwise, they have encountered a lot of uncertainties in determining specificity of various documents. Now, at the point of document design, they have focused on material completeness, narrative balance, as well as the language of the documents. Titles of document sections and text in comments to mind map nodes rather completely characterized future paper. That's when I pointed stylistic features of each document to the students:

- Language of requirement specification should be dry, strong and unambiguous, the requirements themselves are broken up into discrete fragments (for traceability, testing, etc.), which correspond to individual sections of the document.
- Style of architecture specification should be narrative, with examples and explanations.
- User manual allows less formal and more emotional style, the document is always addresses the user and his needs.

Some students tried to create an entire document in Comapping, turning comments into full text. I tried to prevent it, since writing a document is still more comfortable in the text editor.

4.4 Document Development Cycle

In this phase students wrote documents. Transition from mind maps to texts was carried out semi-automatically: Comapping is able to generate rtf-file on mind map putting nodes and their comments into the file. For

development of documents Microsoft Word was used. At this phase the main focus was on mastering Microsoft Word rules (headers, layout features, page numbers, etc.), as well as syntax and grammar. This phase was essentially more lightweight, after Document Design Cycle was introduced to the course. 3-4 iterations were enough for the students to produce the results significantly exceeding the results of previous years. I spent much less effort on revision and discussion of papers.

5. Evaluation, feedback and conclusions

After completing the course in 2011 all of course students (16 persons) filled the evaluation forms, which contained two kinds of questions: answers for the first ones should have been numbers from 5 to 1 (5 = Agree, 1 = Disagree), answers for the second part of the questions should have been written (so called open questions). Average score of answers to the first set of questions concerning documentation training is presented in Table 1.

Table 1.
Student perception of documentation training

Question	Average value
Training value	4,000
Training interest	3,667
Learned skills	3,533
Value of mind maps	3,533
Simplicity of learning and using Comapping	4,850

Students sufficiently evaluated worth of the documentation training. The relatively low degree of interest in the training is due to the fact that, firstly, the students clearly express more interest to code writing, secondly, development of documentation is not very popular among students, and thirdly, students did not quite understand relation of documentation with software development. I feel that writing documentation remained a separate and self-sufficient topic, though they understood it is necessary in software development. By resolving the last argument it is to be hoped that students' enthusiasm pertaining to documents design will also increase. In addition, it is necessary to seek additional ways to make the learning process more interesting. In particular, group assignments should be involved more intensively (e.g. peer reviews of documents). Working in groups have left a great impression on students, but this approach was actively used in other parts of the course, and during documentation development was used to a little degree.

Although students aware of mind maps practicality in documentation development, however, underestimated them. I think that the reason was me wanting to create a contrast and suggesting that they first try to develop documents a common natural way, and then introduced the Document Design Cycle. This preliminary phase was longer than it would have been necessary.

Simplicity and ease with which students have mastered the product Comapping is also worth noting. I spent almost no time explaining, the students made out themselves and were quite versed. In general, it seems that potential of mind maps and Comapping in documentation training had not been fully revealed yet.

Half the students felt that they should continue to develop documentation development skill. 25% indicated that have sufficiently mastered this skill, and 25% — that «these skills, though useful, but mastering them is boring and it is unlikely they will do it in the future, as there are more interesting things». It's important the students recognize real progress in the area.

The presented method, with some modifications, can be used in different contexts. For example, students may by now already be able to write technical documentation, and then they have to be taught specific software engineering documentation only. Or they may already possess the knowledge and skills to develop requirements, software design, etc., and then appropriate training is not needed. In both cases, the relevant parts of the method shall not be used. At last the suggested approach can be used in teaching to write thesis and papers.

It should be noted the mind maps can be used not only for teaching document writing but also for industrial document development. In particular, they may be used for document knowledge management in the software companies presenting knowledge that encapsulated in technical documents. The collection of such mind maps serves as a part of corporate memory or knowledge portal. These mind maps can be converted into ontologies, and a set of such ontologies may be managed automatically, e.g. using the context search engine. Using ontologies as a core of corporate knowledge portals as now a widely used practice [18].

References

[1] E. H. Weiss, How To Write Usable User Documentation (2nd edn, Phoenix: Oryx Press, 1991).

[2] T. T. Barker, Writing software documentation: A task-oriented approach (Part of the Allyn & Bacon Series in Technical Communication, Longman, 2002).

[3] T. Buzan, The mind map book, (2nd edn, BBC Books, London, 1995).

[4] D. Koznov, M. Pliskin, Computer-supported collaborative learning with mind-maps, Communications in Computer and Information Science, 17, Springer, 2008, 478–489.

[5] U. Nikula, O. Gotel, J. Kasurinen, A motivation guided holistic rehabilitation of the first programming Course. Journal ACM Transactions on Computing Education (TOCE), 11 (4), 2011, 24.

[6] C. W. Liew, Teaching software development skills early in the Curriculum through software engineering, ITiCSE, 2005, 133–137.

[7] R. McCauley, U. Jackson, Teaching software engineering early: experiences and results, ACM SIGCSE Bulletin, 31 (2), 1999, 86–91.

[8] J. Santore, T. Lorenzen, Use writing class techniques to create software design documents . June 2009, SIGCSE Bulletin , 41(2), 136–137.

[9] J. Liu, J. Marsaglia, D. Olson, Teaching software engineering to make students ready for the real world, Journal of Computing Sc iences in Colleges, 18 (2), 2002, 43–50.

[10] D. Schwartz, CLOWNS – a software engineering semester project currently in-use in kindergarten classes. Proceedings of the International Conference on Frontiers in Education: Computer Science & Computer Engineering, 2008, 344–349.

[11] G. Xing, Teaching software engineering using open source software, ACM Southeast Regional Conference 2010, 57.

[12] E. Bareisa, E. Karciauskas, E. Macikenas, K. Motiejunas, Research and development of teaching software engineering processes. CompSysTech, 2007, 75.

[13] J. Noble, S. Marshall, S. Marshall, R. Biddle, Less extreme programming. ACE '04: Proceedings of the Sixth Australasian Conference on Computing Education, 2004, 217–226.

[14] K. Beck, Extreme programming explained: embrace Change (2nd edn, Addison-Wesley, 2005).

[15] K. Anewalt, J. A. Polack-Wahl, Teaching an iterative approach with rotating groups in an undergraduate software engineering course. Journal of Computing Sciences in Colleges, 25 (6), 2010, 144–151.

[16] K. Anewalt, Dynamic group management in a software projects course. Journal of Computing Sciences in Colleges , December 2009, 25(2), 146–151.

[17] Alan Cooper, The inmates are running the asylum (Sams Publishing, 2004).

[18] T.A. Gavrilova, H.B.Jin, Ontology-based knowledge portal development for university knowledge management. Proceedings of 4th International Conference on Networked Computing and Advanced Information Management, 2008, 552–559.