Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

An Empirical Investigation of Search-Based Computational Support for Conceptual Software Engineering Design

Christopher L. Simons and Ian C. Parmee
Department of Computer Science,
University of the West of England,
Bristol, BS16 1QY, United Kingdom.
chris.simons@uwe.ac.uk, ian.parmee@uwe.ac.uk

***Abstract*—Conceptual software engineering design is an intensely people-oriented and non-trivial activity, yet current computational tool support is limited. While a number of search-based software engineering approaches to support software design have been reported, few empirical studies into their application have been described. This paper reports the findings of an observational study of conceptual design episodes in a UK higher education problem domain. When compared with a manual design episode, a design episode enabled by a user-interactive, search-based, evolutionary computation tool generates a large number of useful and interesting candidate designs, and provides enhanced qualitative and quantitative evaluation. It is also found that tool-supported visualization of UML class designs offers opportunities for sudden design discovery, and that designers respond positively to opportunities to explore and exploit multiple candidate designs. It appears therefore that search-based computational tool support offers high potential in the support of conceptual software engineering design.**

***Keywords*—evolutionary computation, software design, search, user-interaction.**

I. INTRODUCTION

Conceptual software engineering design is an intensely people-oriented activity wherein concepts and information relating to a relevant problem domain are identified and evaluated. However, the act of conceptual software design is non-trivial and demanding for software engineers to perform. Furthermore, the current extent of computational tool support for conceptual software design is limited. Unified Modeling Language (UML) tools e.g. [1] appear to provide the designer with a means to formally record the output of design decisions rather than support the design process. For any conceptual design support tool to be effective, it must proactively support and enable the design process itself. Following on from previous research [2], [3], we continue to suggest that it is impossible to completely automate the people-oriented richness of the design process and exclude the designer. Rather, we would suggest that it is highly desirable to support (not replace) the designer. For this to be effective, the human designer and the support tool must collaborate interactively in an iterative and opportunistic process beginning with a design problem domain and leading to useful and interesting design solutions.

A number of approaches have arisen in the emerging field of Search-Based Software Engineering (SBSE) to assist the human software engineer with design activities. For example, heuristic search techniques have been suggested by Mitchell and Mancoridis as a mechanism to extract design abstractions from source code [4]. Search-based approaches to module clustering have been reported wherein the architecture of software modules is reorganized with respect to various cohesion and coupling metrics [5]. More recent research reports the automatic reverse engineering of source code to infer subsystem abstractions which may be useful for a variety of software maintenance activities [6]. Search-based refactoring approaches have also been proposed and show promising results [7], [8]. However, it is characteristic of module clustering and refactoring that both are essentially down-stream design activities with respect to the software lifecycle. It seems likely that the software engineer will have already designed, implemented and deployed at least initial versions of the software system before such down-stream search-based approaches may be of practical benefit.

Up-stream search-based design support approaches are not plentiful in the literature, although Lo and Chang [9] report the up-stream application of clustering techniques to software component architecture design, and Aversano *et al.* [10] report a search-based approach to semi-automatically support the design of service compositions by means of genetic programming. Previous work by the authors [11], [12] reports results of search-based support for upstream conceptual design, in which the human designer and computational tool support collaborate to jointly steer the search through a space comprised of a mass of candidate solutions. As an upstream design activity, collaboration between software engineer and computational tool support is considered crucial to enhance the integrity of conceptual software designs.

However, despite the encouraging results achieved, empirical studies of the effectiveness of up-stream search-based support approaches appear lacking in the research literature. Specifically, no examples of empirical studies of search-based computational tool support for industrial conceptual software design are evident. This paper aims to address this shortfall and reports the findings of participant observation which are used to assess the effectiveness of user-

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

interactive search-based tool support for up-stream conceptual software design.

The structure of the remainder of the paper is as follows. Section 2 provides a brief overview of user-centered, evolutionary search-based approach for support of conceptual software design. Section 3 describes the investigation problem domain, while section 4 details the method used to assess the participant observation. Section 5 specifies observation and data collection, section 6 details the results obtained and section 7 offers analysis and discussion of the results. Threats to validity are presented in section 8, and the paper concludes in section 9.

II. USER-CENTERED, EVOLUTIONARY SEARCH

The evolutionary search approach evaluated in this paper is described previously at [11]. In overview, the evolutionary search incorporates representations of both problem and solution spaces which are manipulated by a multi-objective Non-Dominated Sorting Algorithm inspired by Deb's NSGA-II [13]. The problem space representation is derived from the application of use cases [14], which are well understood and widely applied in software engineering requirements capture. Actions and data are identified from the narrative scenario text of use cases relevant to the problem domain under study. Thus the problem space is represented by a set of actions and a set of data derived directly from the requirements of the problem domain. The solution space is represented as UML classes, which serve as placeholders for attributes and methods. The solution space is comprised of a set of attributes and a set of methods, which are partitioned among the UML classes. The set of solution attributes are directly derived from the problem space set of data, while the set of solution methods are directly derived form the problem set of actions. Such derivation provides inherent traceability from the problem space to the solution space.

Acting upon the solution space, optimization and diversity preservation operators are inspired by the elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II) proposed by Deb [13]. In NSGA-II, an offspring population of the same size is created from the parent population. Using cohesion and coupling metrics as fitness functions, the combined population is non-domination sorted into ordered 'fronts' of equivalent optimality. The new population is filled by solutions of different non-dominated fronts, one at a time: the filling starts with the best front, and continues with the next best and so on until the population size is met or exceeded. Solutions that do not make the new population are discarded. In the early stages of evolution, many fronts are evident in the population. However, as evolution proceeds, the number of fronts decreases until a small number of fronts, including the Pareto-optimal front, remains.

Designer preferences and software agents enable joint human-computer collaborative interaction during search of the solution space [12]. Indeed, interactive, joint human-computer activity appears to map well to the notion of a conceptual software design episode. Design episodes have been observed in many fields of design (e.g. [15]) and software development (e.g. [16]). Design episodes provide a useful vehicle for human-computer interaction wherein human engineers and

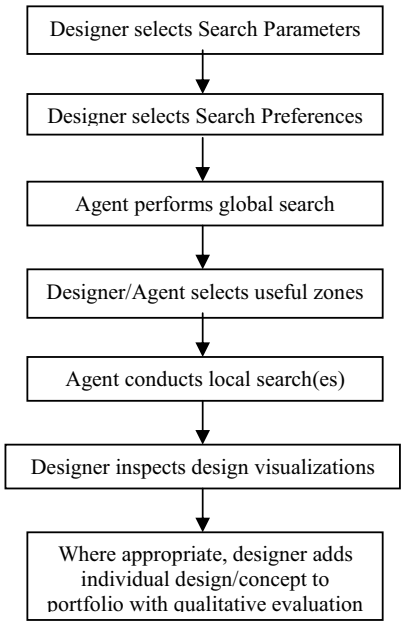

Figure 1. Flow chart of design episode

software agents progressively focus on increasing superior design concepts, providing a natural and effective way of narrowing the population based search.

A search-based conceptual software design episode begins with multi-objective search of the global solution space using metrics and preferences chosen by the software engineer in conjunction with agent-based utility values. A flow chart summarizing the components of a design episode is shown in fig. 1. In reality the flow of design is iterative; fig. 1 shows a sequential process for simplicity. Multi-objective search is performed by a Global Search Agent, which collaborates with a Concept Isolator Agent to halt global search at an optimum point at which local zones of particular software design concepts are emerging. Thereafter, the episode may progress to a number of local searches (one in each emerging zone) conducted concurrently by Local Search Agents. As in global search, local search is guided jointly by designer preferences and agent search utility values. At any point in the search, the software engineer may visualize a candidate UML class diagram solution. Visualization reveals color coding of cohesion values of individual class diagrams - highly cohesive classes are red for 'hot', classes of low cohesion are blue for 'cool'. Having identified interesting and useful UML class diagram solutions, the software engineer may place them in the episode portfolio together with reasons why the class diagram is retained. In addition, in the course of a design episode, an Event Logger Agent records significant events to an episode log, which enables a detailed record of the history of the episode.

III. PROBLEM DOMAIN

The problem domain chosen for investigation is the specification of an extension to a student administration system performed by the in-house information systems department at

the University of the West of England, UK. Over recent years, this university has sought to record and manage outcomes relating to personal student development during their studies. A strategic decision was then made to extend the capabilities of the existing student administration system to be able to record and track students' personal development in parallel with their academic achievement.

In line with standard practice for the in-house information systems department initial requirements capture activities have involved regular, highly iterative and people intensive, interactive sessions with stakeholders where 'mock-up' scenarios of usage have been piloted under conference room conditions. While no specific development methodology has been employed, principles common to DSDM [17] and agile methods [18] predominate. During the interactive pilot sessions, no computational tool support has been deployed except for rapid construction of mock graphical user interfaces (GUIs). The pilot sessions successfully identified system actors and four main goals that the actors would wish to achieve within a scenario of interaction with the system. The four goals included:

- the ability to record a personal development outcome for an individual student;
- the ability to record personal development outcomes for a batch of many students;
- the ability to generate various reports on personal development outcomes; and
- the ability to export report results in a format capable of being read in desktop spreadsheet applications.

The four goals have been recorded as use cases and are available at [19].

IV. Method and Case Study Design

The method employed in the empirical study is to observe two conceptual software design episodes; one with search-based computational tool support and one without. The manual episode is a baseline against which comparisons and contrasts with the search-based tool support episode may then be drawn.

The software engineering participants being observed included a project manager and business analyst who work within the in-house information systems department under investigation. The project manager and business analyst have been selected for observation as they typically perform conceptual software design within the in-house information systems department. The project manager has a bachelor's degree in systems analysis and 20 years professional experience of requirements capture, analysis, design and project management of information systems. The business analyst has bachelor's degree in business information systems and 7 years professional experience of requirement capture, analysis and design of information systems. Two issues arise at this point:

- How generalizeable might be the results when the number of participants is small?
- How representative is this sample of the larger population of software engineers?

Given the relative lack of empirical studies reported in the literature for search-based engineering, it is hard to answer this question. There exists little or no population data to compare this sample against, and there is no standard type of individual who performs conceptual software design – education, professional experience, job context and competencies may differ markedly. However, the two individuals selected are held in high esteem by their colleagues, and are representative of some segment of the population of software engineers who perform conceptual software design.

The method of the investigation compares and contrasts two design episodes, based on the same problem domain. Clearly, a higher degree of confidence in observations would have been achieved from observing further design episodes. Unfortunately, finding suitable people-intensive industrial design situations appropriate to observational studies is not a trivial task. With respect to method, the manual episode has been conducted first, followed by the tool supported episode. Data obtained from the first episode is therefore treated as a baseline for comparison with the second.

Visual and audio recordings have been made for both design episodes, and a textual transcript of verbal utterances has been taken from the recordings. One of the authors has been present at both design episodes in order to produce recordings but has remained silent and non-participatory throughout, except with respect to the physical mechanics of producing recordings and tool support in the second episode.

V. Observation and Data Collection

Measurements have been selected to investigate the richness of the design episodes both in terms of outputs produced and the means by which the outputs are produced. With respect to the means by which the outputs are produced, a number of characteristics of interaction have been investigated including approaches to:

- concept generation;
- iteration;
- opportunistic realization; and
- medium of interaction.

According to Liu *et al., "conceptual design should contain two types of steps: divergent in which alternative concepts are generated, and convergent in which these are evaluated and selected."* [20]. The suggestion of Liu *et al.* is consistent with reports within software engineering by Glass, who suggests that design is a complex, iterative process in which initial designs are usually wrong and certainly not optimal [21]. Thus divergent and convergent design activities have been observed and recorded as a measure of the richness of the design episode.

Iteration is widely regarded as a necessary and natural component of design (e.g. [21], [22]) and so iteration between not only the problem and solution spaces, but also convergent and divergent design activities have been observed.

Furthermore, sudden discovery moments and opportunistic understandings (e.g. [21], [23]) have been noted as being significant events within design episodes and so these have been observed too.

Finally, as an indicator of the richness of the design episode, the medium of interaction between the two designers has been observed, be it verbal, paper-based sketching, interacting via the search-based support tool, or via UML class modeling.

Textual transcripts of the two episodes have been analyzed according to design mode, design activity, and the occurrence of design events. Design modes and design activities have been analyzed within 20 second intervals in the design episode. 20 second intervals have been chosen to provide a reasonable level of granularity of analysis. Design modes include:

- Space – is the design episode focused primarily on the problem or solution space in each 20 second timed interval?
- Thrust – is the thrust of the design episode primarily convergent or divergent in each 20 second time interval?
- Medium – is the medium of designer interaction verbal, sketching, search-based tool supported, or UML class modeling in each 20 second time interval?

Design Activities include:

- Evaluation – are the designers primarily evaluating individual candidate designs in a 20 second time interval?
- Generation – are the designers primarily generating candidate designs in a 20 second time interval?
- Trading-off - are the designers primarily trading-off between multiple candidate designs in a 20 second time interval?
- Scoping – are the designers primarily considering if a candidate design is in scope during a 20 second time interval?
- Reflective silence pauses – have the designers paused for silent reflection?

Design Events are discrete happenings at a point in time in the design episode and include:

- Request for clarification – a designer requests a clarification of design activities of the other;
- Explanation of understanding – a designer explains their understanding of a design activity to the other;
- Sudden discovery – a designer expresses an "ah-ha!" moment of sudden discovery of a design concept or design concept relationship;
- Realization of constraint – a designer expresses a moment of realization that a candidate solution is constrained in some manner by the problem domain requirements;
- Realization of inferred requirement – a designer expresses an insight of an inferred requirement i.e. although not explicitly stated in the case study problem domain specification, a further requirement is inferred as consistent with the specification;
- Inspection of a candidate UML class diagram – a designer inspects a candidate UML class diagram.
- Add a UML class diagram to portfolio – a designer adds a useful and interesting UML class diagram to the episode portfolio.

VI. RESULTS

A. Duration

The duration of the baseline manual conceptual design episode was 37 minutes and 2 seconds (2122 seconds), while the duration of the test design episode with search-based tool support was 55 minutes and 23 seconds (3323 seconds). A textual transcript of the baseline manual conceptual design episode is available from [24]; a textual transcript of the test design episode with search-based tool support is available from [25].

B. Designs Produced

No design artifacts of conceptual software designs were produced during the baseline manual conceptual design episode. While much verbal interaction centered on the explanation of the concept of "Student" and its associated information, no drawings or UML diagrams were produced.

Many conceptual designs were produced in the course of the search-based tool supported design episode. Analysis of the transcript reveals that 30 candidate class diagrams were inspected, and from these, 7 were added to the portfolio with the search-based support tool. While adding candidate class diagrams to the portfolio, the two designers recognized a "Student" concept after 5 minutes, an "Award" concept after 6 minutes, a "Report" concept after 16 minutes, and a "Rule" and a "Development" concept after 23 minutes. Thus in total, 5 concepts were identified in the search-based tool supported design episode, which contrasts with one concept identified in the manual design episode.

Designer reaction to the introduction of the search-based computational tool was positive. After a period of familiarization, the two designers became fluent in the use of features provided by the search-based tool. Indeed, by the end of the tool supported design episode, both designers were freely suggesting useful enhancements and extensions to the search-based tool, which the authors plan to incorporate into future research.

C. Richness of Design Episodes

A summary of all observation data is shown in Table 1. Where proportions are reported for episode modes and activities, these relate to the proportion of the mode or activity as a part of the total duration of the episode. Where average frequencies are reported for episode events, the average frequency in seconds is the episode duration divided by the number of events.

TABLE 1. Observation Data

ASPECT		OBSERVATION	BASELINE	*Proportion*	TEST	*Proportion*
Duration		Seconds	2212		3323	
		Minutes - Seconds	37-02		56-23	
Mode	Space	Problem	34	0.307	4	0.024
		Solution	77	0.699	158	0.950
	Thrust	Convergent	45	0.406	87	0.523
		Divergent	17	0.153	37	0.222
		Iterations	10		28	
	Medium	Verbal	110	0.994	10	0.060
		Sketching	1	0.090	0	0.000
		Tool interaction	0	0.000	71	0.427
		UML Class Modeling	0	0.000	85	0.512
Activity		Evaluation	60	0.542	67	0.403
		Generation	4	0.036	28	0.168
		Trading-off	5	0.045	36	0.216
		Scoping	1	0.009	0	0.000
		Reflective silence	0	0.000	9	0.054
				(Ave Freq)		*(Ave Freq)*
Events		Request for Clarification	37	59.780	32	103.840
		Explain Understanding	41	53.950	37	89.910
		Sudden Discovery	2	1106.000	18	184.610
		Constraint realization	3	737.330	0	
		Inferred Requirement	3	737.330	0	
		Inspect candidate	0		30	110.770
		Add to portfolio	0		7	474.710

VII. ANALYSIS

With regard to the duration of the two episodes, the participants appeared to respond positively to opportunities presented to explore and exploit designs, resulting in more time spent in the test episode than the manual. Indeed, the test episode would have continued longer had not the tool encountered an out-of-memory problem.

With respect to design modes observed, it is clear that iteration between the problem and solution spaces is richer in the manual design episode; less problem / solution iteration is evident in the tool supported episode. This suggests that tool support tends to focus the designers on the solution space, which may hinder potential problem reformulation. The design thrust of the manual design episode is essentially convergent whereas the tool supported episode shows more divergence and iteration. This may be due to population-based search providing superior exploratory support. The medium of the manual design episode shows dramatic differences to the medium of the tool supported episode. The manual design episode is highly verbal, with occasional paper sketching of graphical interfaces but no UML modeling. However, the tool supported design episode is greatly more productive in terms of UML modeling, with over one half of the episode focused on this.

Within the design activities, evaluation is observed to be the dominant activity in the manual design episode. Indeed, with few candidate designs being generated, qualitative evaluation appears to dominate. However, the population-based search tool provides much designer support in generating candidate designs. Interestingly, tool support also provided opportunities for both quantitative and qualitative evaluation, as well as trade-off evaluation. Trade-off evaluation appears to be a difficult activity in the manual design episode as it requires designers to remember many designs for comparison. Conversely, in the tool supported episode, a design portfolio is provided which greatly assists trade-off evaluation – a significant benefit of the design support tool. Furthermore, it is observed that visualization of UML class models enables cognitive reflection, which stimulates the designers. Nine reflective periods of silence were observed in the tool supported episode whereas none were observed in the manual design episode.

Regarding design events, designers were observed to make more requests for clarification at a greater frequency in the manual design episode. In addition, a greater number of verbal explanations of understanding were observed in the manual design episode. This is consistent with the highly verbal medium in which the manual design episode is conducted. Conversely, requests for clarification and explanations were less abundant in the tool supported episode; it seems likely that this is due to the visualizations of candidate design solutions that promoted shared understanding of the designs. It is significant that the number of sudden design discovery events were observed to be higher in the tool supported episode (18) than in the manual design episode (2). This finding appears to be consistent with the nine periods of reflective silence observed in tool supported episode. It seems likely that rich

generation of alternative candidate designs, when combined with opportunities for visual reflection, affords more opportunities for moments of sudden design discovery. Lastly, it is also significant that in the tool supported episode, unique candidate designs were inspected by the designers on 30 occasions; i.e. a candidate design was inspected roughly once every two minutes. The designers having been stimulated by the visualization on the UML designs, 7 were added to the portfolio.

VIII. THREATS TO VALIDITY

Two designers have been observed in the course of this empirical investigation. While a greater number of designers would add weight to the investigation, this situation reflected the reality of the team under study. Moreover, the two designers are representative of some section of the software engineering community where empirical investigations available in the literature are few.

The above analysis is also tempered by the fact that the same problem domain has been used for both episodes. Given that the manual design episode has been conducted firstly, the designers will take any acquired knowledge of the problem domain into the second, tool supported episode. Given this learning effect, it might be reasonably expected that the designers would arrive at a greater number of designs in the tool supported episode. However, it is argued that the number of trade-off evaluations, moments of sudden design discovery, and candidate inspection and additions to the portfolio is significantly higher in the tool supported episode, even accounting for any learning effect.

IX. CONCLUSIONS

Analysis of the observation data reveals that in this small scale empirical investigation, interactive, search-based tool support for conceptual software design could be considered effective at generating multiple candidate design solutions, and highly productive in terms of UML class models. In the manual design episode, as few candidate designs are generated, qualitative evaluation of candidate designs is the dominant design activity. In contrast, in the search-based tool supported design episode, (i) generation of candidate designs is more balanced with evaluation, (ii) evaluation is both qualitative and quantitative and (iii) trade-off analysis is greatly enhanced. Furthermore, visualization of UML class designs, when combined with generation of multiple candidate designs, enables periods of reflection that offer opportunities for sudden design discovery. In addition, observations suggest that designers respond positively to opportunities presented to them to explore and exploit multiple candidate designs. We therefore conclude that search-based computational tool support offers high potential in assisting conceptual software engineering design and on this basis we are continuing with research involving empirical studies into larger industrial scale case studies of search-based computational tool support.

REFERENCES

[1] Object Management Group, UML vendor directory listing [Online]. Available: http://uml-directory.omg.org/vendor.list.htm.

[2] I.C. Parmee, A. Watson, D. Cvetkovic, C.R. Bonham, "Multi-objective satisfaction within an Interactive Evolutionary Design Environment", *J. Evol. Comput.*, vol. 8, no. 2, 2000, pp 197–222.

[3] D. Cvetkovoc, I.C. Parmee, "Agent-Based support within an Interactive Evolutionary Design System", *Artif. Intll. Eng. Des. Anal. Manuf.*, vol. 16, no. 5, 2002, pp. 331-342.

[4] B.S. Mitchell, S. Mancoridis, "Using heuristic search techniques to extract design abstractions from source code", in *Proc. Genetic and Evol. Comput. Conf. (GECCO 2002)*, New York, USA, 2002, pp. 1375-1382.

[5] O. Magbool, H.A. Babri, "Hierarchical clustering for software architecture recovery", *IEEE Trans. Softw. Eng.,* vol. 33, no. 11, 2007, pp. 759-780.

[6] B.S. Mitchell, S. Mancoridis, "On the evaluation of the Bunch search-based software modularization algorithm", *Soft Comput.*, vol. 12, no. 1, 2008, pp. 77-92.

[7] M. Harman, L. Tratt, "Pareto optimal search-based refactoring at the design level", in *Proc. Genetic and Evol. Comput. Conf. (GECCO 2007)*, London, UK, 2007, pp. 1106-113.

[8] M. O'Keeffe, M.O. Conneide, "Search-based refactoring for software maintenance", *J. Sys. Softw.*, vol. 81, no. 4, 2008, pp. 502-516.

[9] S.-C. Lo, J.-H. Chang, "Application of clustering techniques to software component architecture design", *Intl. J. Softw. Eng. Know. Eng.*, vol. 14, no. 4, 2004, pp.429-439.

[10] L. Aversano, M. Di Penta, K. Tanejy, "A Genetic Programming approach to support the design of service compositions", *Comp. Sys. Sci. Eng.*, vol. 21, no. 4, 2006, pp 247-254.

[11] C.L. Simons, I.C. Parmee, "A cross-disciplinary technology transfer for search-based evolutionary computing: from engineering design to software engineering design", *Eng. Opt.*, vol. 39, no. 5, 2007, pp. 631-648.

[12] C.L. Simons, I.C. Parmee, "User-centered, evolutionary search in conceptual software design", in *Proc. IEEE Congr. Evol. Comput. (CEC '08) (IEEE World Congr. Comput. Intell.)*, Hong Kong, 2008, pp.869-876.

[13] K. Deb, *Multi-objective optimization using evolutionary algorithms*, Wiley, 2001.

[14] I. Jacobsen, M. Christerson, P. Jonsson, G. Overgaard, *Object-oriented software engineering – a use case driven approach*, Addison-Wesley, 1992.

[15] B. Lawson, *What designers know*, Architectural Press (Elsevier), 2004, pp. 17-18.

[16] K. Beck, "A development episode", Chapter 2 in *Extreme programming: embrace change,* Addison-Wesley, 2000.

[17] Dynamic Systems Development (DSDM) Consortium, [Online]. Available: http://www.dsdm.org/.

[18] Agile Alliance, [Online]. Available: http://www.agilealliance.org/.

[19] Use case descriptions for Student Development Program [Online]. Available: http://www.bit.uwe.ac.uk/~clsimons/CaseStudies/CaseStudyUseCases.pdf

[20] Y.-C. Liu, T. Bligh, A. Chakrabati, "Towards an 'ideal' approach for Concept Generation", *Design Studies*, vol. 24, no. 4, 2003, pp. 341-355.

[21] R.L. Glass, "Facts and fallacies of Software Engineering", Addison-Wesley, 2003, pp. 79-83.

[22] Jin, Y., Chusilp. P., "A study of mental iteration in different design situations", *Design Studies*, vol, 27, no. 1, 2006, pp. 25-55.

[23] R. Guindon, "Designing the design process: exploiting opportunistic thoughts", *Hum.-Comput. Interact.*, vol. 5, no. 2-3, 1990, pp. 305-344.

[24] Transcript of baseline manual conceptual design episode [Online]. Available: http://www.bit.uwe.ac.uk/~clsimons/CaseStudies/BaselineTranscript.pdf

[25] Transcript of test conceptual design episode with search-based tool support [Online]. Available: http://www.bit.uwe.ac.uk/~clsimons/CaseStudies/TestTranscript.pdf.