How Design Style Relates to the Representational Power of Design Outcomes

Cláudia Werner, Rafael Cepeda, Marcelo Schots
Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
{werner, rcepeda, schots}@cos.ufrj.br

Leonardo Murta
Instituto de Computação
Universidade Federal Fluminense
Niterói, Brazil
leomurta@ic.uff.br

Abstract — **Our hypothesis in this paper is that the design style can relate to the representational power of the design outcomes. To study this hypothesis, we analyzed three professional design sessions showing people designing software with different strategies (*i.e.*, different design styles). We also analyzed the design outcomes provided by each strategy and the representational power of these design outcomes. Our results provide some evidence in terms of two metrics: innovation and coverage. Finally, we discuss some automation requirements for tools that can support the design process. One of the conclusions we extract from this study is that a unified method for designing software, one that combines tools and techniques found on an object-oriented design style with tools and techniques found on a *Human-Computer Interaction* (HCI) design style could lead to more powerful design outcomes. This, in turn, could lead to more innovative and effective products that fullfil the requirements imposed by the user's problem.**

Keywords - software design; professional design; design styles; design outcomes; design tools

I. Introduction

Software design is a problem-solving activity that links the problem domain with the solution domain [1]. In other words, it could be thought as a process that transforms the output of the requirements analysis phase into a formal or informal specification of the solution, which serves as input to programmers. Usually, this transformation process is conducted by some abstraction mechanism, *i.e.*, some process of removing detail to simplify and focus attention along with the process of generalization to identify the common core or essence of the object under analysis [2]. The essence of this is identifying, organizing, and presenting appropriate details while suppressing details that are not currently relevant [3].

The strategies adopted during design, called design styles in this paper, may dictate the way design outcomes are structured. These design outcomes, on the other hand, may have different representational power. In some cases, fewer design outcomes are able to represent many aspects (views) of the software (architecture). In other cases, each design outcome is focused on specific aspects of the software, requiring complementary design outcomes to allow a comprehensive view of the software.

In this paper we analyze three videos of professional design sessions according to different design strategies. Each of these design sessions has its particular characteristics, differing both in design style and design outcomes. Our main goal is to analyze how the three different design styles lead to different design outcomes. It is important to note that our conclusions were strictly derived from the videos. We did not have access to direct interview or asked questions to the participants.

In the video sessions, each professional team was asked to design a traffic flow simulation program. One of the expected results of this task is to present the achieved design to a team of software developers who should be able to implement it. The traffic flow simulation program aims to help civil engineering students to understand the basic concepts and theory regarding the topic of traffic signal timing, allowing students to learn from practice. Along with other requirements, students must be able to create a visual map of an area and layout roads in some chosen pattern. Besides, the students must also be able to describe the behavior of the traffic lights at each intersection. Therefore, as it can be observed, this program has a very important subject on user interaction to deal with.

The first design session (called DS1) team was composed by two men with a noticeable difference in age. The older member (38 years old) has 16 years of professional experience, while the younger member has almost 11 years of professional experience, as mentioned in the video. They both have a strong background on object-oriented software projects. The second design session (called DS2) team has one man and one woman as members. Both appeared to have around the same age, but she has a longer time of professional experience, demonstrated by her 26 years working with object-oriented systems, user interface (UI) design, and application development. The other member has 20 years of experience, most on interaction design and on curriculum development fields. The third design session (called DS3) team has two men as members. The difference on age between them seems to be at most 10 years. However, their experience could not be detected based on the video analysis.

Our analysis is presented in this paper according to the following process:

1. Identify the design style of each design session
2. Identify the type of outcome of each design session
3. Identify the time spent to build each outcome type identified in step 2

4. Identify appropriate metrics to compare the representational power of the outcomes identified in step 2

5. Contrast the representational power of the outcomes identified in step 2, considering the time spent to build them, identified in step 3, and the metrics identified in step 4

6. Provide a qualitative analysis of the design sessions to explain the finding of the quantitative analysis performed in step 5, relating the design styles to the representational power of the design outcomes and identifying the weak aspects of each design session

7. Suggest some tool requirements to support each of the three design styles according to the aspects identified in step 6

The enactment of this process required us to analyze the whole design sessions data, both in high level, to identify the design styles, and in detail, to measure the outcomes. Moreover, our research considers the existing design styles (*e.g.,* traditional object-oriented design), the existing design outcome types (*e.g.,* class diagrams), and the existing metrics to correlate design style with representational power of design outcomes (*e.g.*, coverage).

The remaining of this paper is divided into 5 sections. Section 2 classifies the design sessions in terms of design styles and design outcomes (steps 1 and 2 of the analysis process). Section 3 provides a quantitative analysis of the design sessions considering specific analysis metrics (steps 3, 4 and 5 of the analysis process). Section 4 provides a qualitative analysis of the design sessions built upon the quantitative analysis described in Section 3 (step 6 of the analysis process). Section 5 provides some insights on how tools could help in the three design sessions (step 7 of the analysis process). Finally, section 6 concludes the paper highlighting its contributions and limitations, and anticipating some future works.

II. Design Sessions Classification

In order to proceed with our analysis, we first identified each design style applied on each design session. Thereafter we analyzed the outcomes produced in each design session. Finally, we mapped the design strategy and the artifacts created during each session.

A. *Design Styles*

A design style can be characterized by its mode of operation and its means of expression [4]. Basically, the mode of operation can be divided in analytical and experimental. The mode of operation can be thought as the process of transforming the available information about the system being developed in appropriate design decisions to accomplish its construction. When abstraction is adopted to simplify the domain information, resulting in a common and deeper understanding, it is said that the analytical operation mode is used. However, if previous experience is used to produce additional information, it is said that the experimental operation mode is used. On the other hand, the means of expression are comprised of specifications and prototypes. The means of expression can be considered as the way development teams express themselves to transform abstract ideas of the software into concrete design artifacts. When formalisms are used to represent different aspects of the initial ideas of the software, the specification expression is in place. However, when developers immediately start to write proof of concept user interfaces, the prototype expression is appropriate.

DS1 concentrates basically on domain analysis in order to build the software design. Based on an informal sketching, the engineers tried to understand the basic dynamics regarding the traffic problem. However, using some key user scenarios, they produced a detailed UML-like design as outcome, containing the most important domain concepts and their relationships. On the other hand, DS2 was guided by the user interaction analysis, finding out the key inherent software concepts and building a common understanding of the problem. As a result, besides creating a high-level design using the previously identified concepts, they also ended up with a user interface (UI) draft of what could be the target application. Finally, DS3 primarily focused on the data analysis associated to the given problem. They used a creative process similar to the process used by the DS1 team. However, in this case, they first identified the concepts, and then performed a domain study, based on a sketching process, to verify its validity. In the end, they were able to produce an informal design, using a free notation, containing the main concepts found, their properties and some associative relationships. The summarization of the design styles applied by each design session can be observed in **Table 1**.

Table 1. Design style summary for each design session

Design Session	Mode of Operation	Means of Expression
1	Analytical	Specification
2	Analytical	Prototype
3	Analytical	Specification

B. *Design Outcomes*

A design outcome can be seen as any artifact produced during the software design phase. We can divide design outcomes into two groups: formal and informal outcomes. The formal outcomes usually follow a specific notation with a strong semantic. They are usually well understood by the majority of software developers. On the other hand, the informal outcomes are notation free, but they have a strong representational power. In this case, developers should previously agree on the meaning of the outcomes via tacit or explicit (*e.g.,* legend) communication. Examples of formal outcomes are UML diagrams, design patterns, business rules, etc. Examples of informal outcomes are diagrams without following a specific notation, usage scenarios, UI prototypes, etc.

DS1 produced the following outcomes: i) A high level UML design containing the main concepts identified during the session and the relationships between them. It is important to note that these relationships do not have a strong semantic. In other words, they can assume in some cases the identity of association and in other cases the identity of composition, for instance. ii) Along with concepts identified before, the team also uses the *Visitor* design pattern [5] as a design decision in order to decouple the controller entity from the remaining structure of the system. iii) The draft made by them of a real scenario represents the problem domain. This scenario, along with others, was used to understand the given domain and the problems related, and to identify the main objects needed to assemble the system. iv) A set of state transition rules was specified to control each possible state to which the system can change.

DS2 resulted in a high level design, containing some UI drafts of what the final system would be, indicating different views of the system in terms of user interactions. These prototypes are linked with arrows that represent the system workflow, like a UI navigation diagram. The design outcome also contains some domain entities (contained in a UML-like diagram), derived from storyboards that were used during design.

DS3 produced a data-driven result, which was similar to a mind-map. It contains concepts (or entities) and their relationships, including the properties of some of these entities. There is also a draft that seemed to be used for understanding the problem domain. The members used different colors for representing different layers to keep track of the design steps. The summarization of the design outcomes derived by each design session can be observed in **Table 2** (note that the designation of formal outcomes does not imply in correctness of notation use).

Table 2. Design outcomes summary for each design session

Design Session	Formal outcomes	Informal outcomes
1	UML diagrams Visitor design pattern State transition rules	Relationships among concepts Draft usage scenario
2	UML diagrams	Requirements refinement UI navigation diagram (including UI drafts)
3	--	Concepts (entities) Relationship among concepts Draft usage scenario

C. *Classification in Terms of Design Styles and Outcomes*

According to our analysis, summarized in Table 1 and Table 2, we could note that the DS1 team followed an analytical operation mode, mainly focusing on refining abstract specifications of the software. Due to that design style, the main obtained outcomes were formal, such as UML diagrams, design patterns and state transition rules. However, we could also observe some informal outcomes, such as relationships among concepts and draft usage scenario, which help on understanding the formal ones.

On the other hand, the DS2 team followed another design style: the use of prototypes as means of abstraction. This team also employed an analytical mode of operation, but focused from the beginning on a proof-of-concept prototype. Note that the concept of prototype here refers to the fundamental idea involving this popular software engineering technique, which is to quickly create results that somehow provide the stakeholders a way to contribute with important feedback about the ongoing development of the product [6]. As a result of the adoption of this style by DS2, we could observe a more comprehensive set of informal outcomes, such as UI drafts and UI navigation diagrams. From these informal outcomes, we could also observe the derivation of one formal outcome, a UML-like class diagram.

Finally, the DS3 team adopted a design style similar to DS1: the use of an analytical mode of operation based on the refinement of specifications as means of expression. However, this team worked in a less systematic way if compared to DS1. As a result, we could observe only some concepts, the relationships among them, and a draft usage scenario as informal outcomes. We did not recognize any formal outcome for this session.

III. Quantitative Analysis

Aiming at comprehending the effect of design styles in the design outcomes, we continue our analysis by adding some quantitative measures in the comparison. The variables considered at this moment are design session duration and number of distinguishable concepts discovered by each session. Initially, it is possible to note that the outcome produced by the DS1 team was built in approximately 1 hour and 53 minutes, and had 7 software concepts. DS2 took roughly 1 hour and 54 minutes and produced, without counting the UI draft, a software design containing 13 software concepts. Finally, the DS3 team spent 58 minutes to create a design with 9 concepts. Note that we are only considering the concepts that appeared in the final model, and not those speculated during the sessions.

Figure 1 shows a comparative view of the three sessions in terms of the outcomes of each design and the time spent on each session.

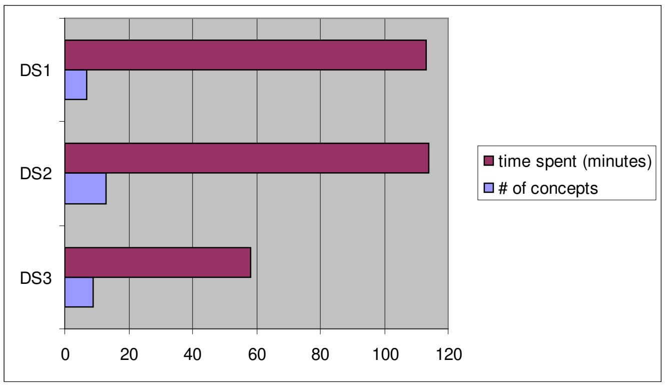

Figure 1. Time spent vs number of concepts: comparative view

The remaining of this section goes deeper on the quantitative analysis, considering additional metrics and providing our considerations regarding the obtained results.

A. Analysis Metrics and Data

By analyzing the commonalities and differences between each element set produced by each team, it is possible to assess the completeness of each design created (when compared to the other two), assuming that every element is relevant to the problem. Note that it is difficult to do a different level of analysis, considering that there is no reference design available and a design inspection was not conducted. Nevertheless, this is not the purpose of this paper.

Regarding the metrics adopted in this quantitative analysis, we were inspired in two classic metrics from the information retrieval field [7]: *precision* (the fraction of retrieved documents which are known to be relevant) and *recall* (the fraction of known relevant documents which were effectively retrieved). However, in our context, a good design session should be able to discover as much concepts as the other design sessions (high coverage level), and also discover some additional concepts that were not discovered in the other design sessions (high innovation level).

Due to the absence of a reference design, we consider the concepts discovered by the other two sessions as reference and apply two metrics: *innovation* and *coverage*. *Innovation* is inspired in *precision* (*precision* has no useful meaning in our context, assuming that all concepts are considered relevant), but represents the fraction of concepts discovered in the design session over analysis that were not discovered in the other design sessions. On the other hand, *coverage* is identical to *recall*, representing the fraction of concepts discovered in the other design sessions that were also discovered in the design session over analysis. For instance, assuming that we are analyzing DS1 against DS2 and DS3, we can state that *innovation* represents the percentage of concepts discovered by DS1 that were not discovered by DS2 and DS3, and *coverage* represents the percentage of concepts discovered by DS2 and DS3 that were also discovered by DS1. In both cases, the denominator of the fraction is the total number of concepts discovered in the other design sessions (in the previous example, DS2 and DS3).

Figures 2, 3, and 4 show the outcomes of DS1, DS2, and DS3, respectively (some image processing operations were applied for showing only the relevant part of each video screenshot). Initially, we can observe the concepts discovered by each design session. First, regarding DS1, the following concepts were discovered: Cop, Car, Intersection, Light, Network, Queue, and Road. On the other hand, DS2 discovered the following concepts: Approach, Block, Car, Simulation Configuration, Intersection, Light, Map, Light Configuration, Road, Sensor, Left-Hand Signal, Traffic Configuration and Simulation Result. Finally, DS3 discovered the following concepts: Controller, Car, Intersection, Map, Road, Signal, Speed, Inputs, and Output. It is important to mention that these are the concepts that we could identify during our analysis. Because of the camera position, environment lighting and the colors used during design, some parts of the design outcomes (especially DS3 outcome) are illegible in the video.

Some concepts that have different names may have the same meaning. Due to this, we provide in **Table 3** a diff among the sets of concepts discovered by the three design session teams. In this table, it is possible to notice that some concepts such as Clock, Simulation, and Time actually correspond to the same intention: provide a way of enacting the traffic model. This table is the main product of this subsection, providing the necessary information to the quantitative analysis itself, presented in the next subsection.

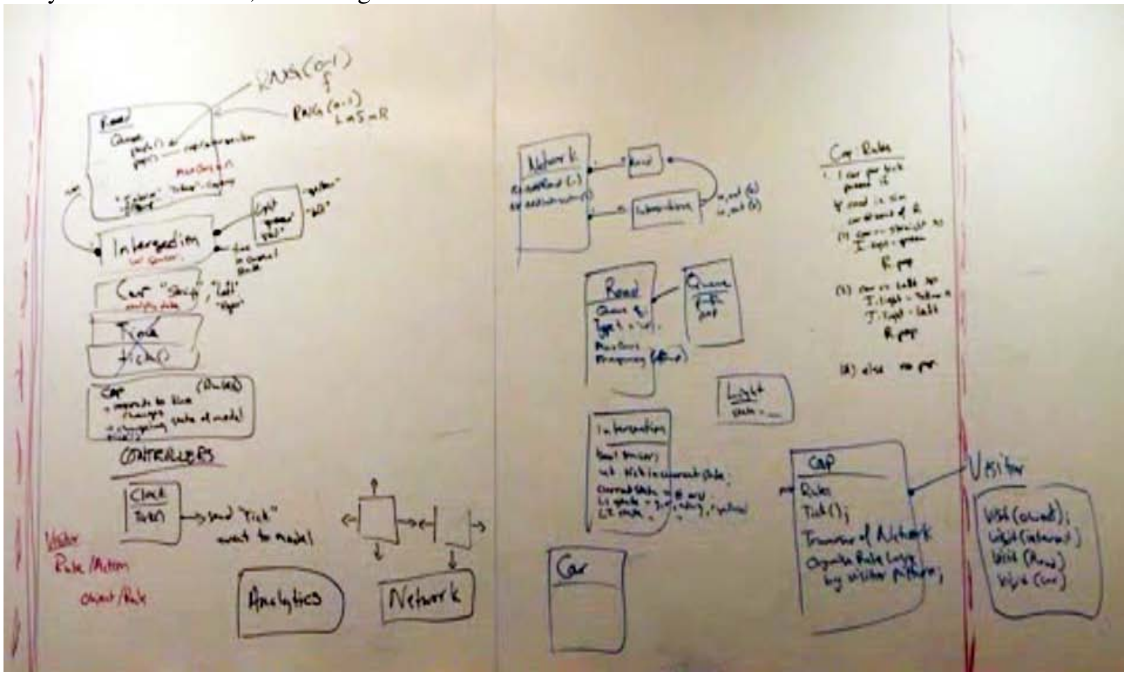

Figure 2. DS1 outcome

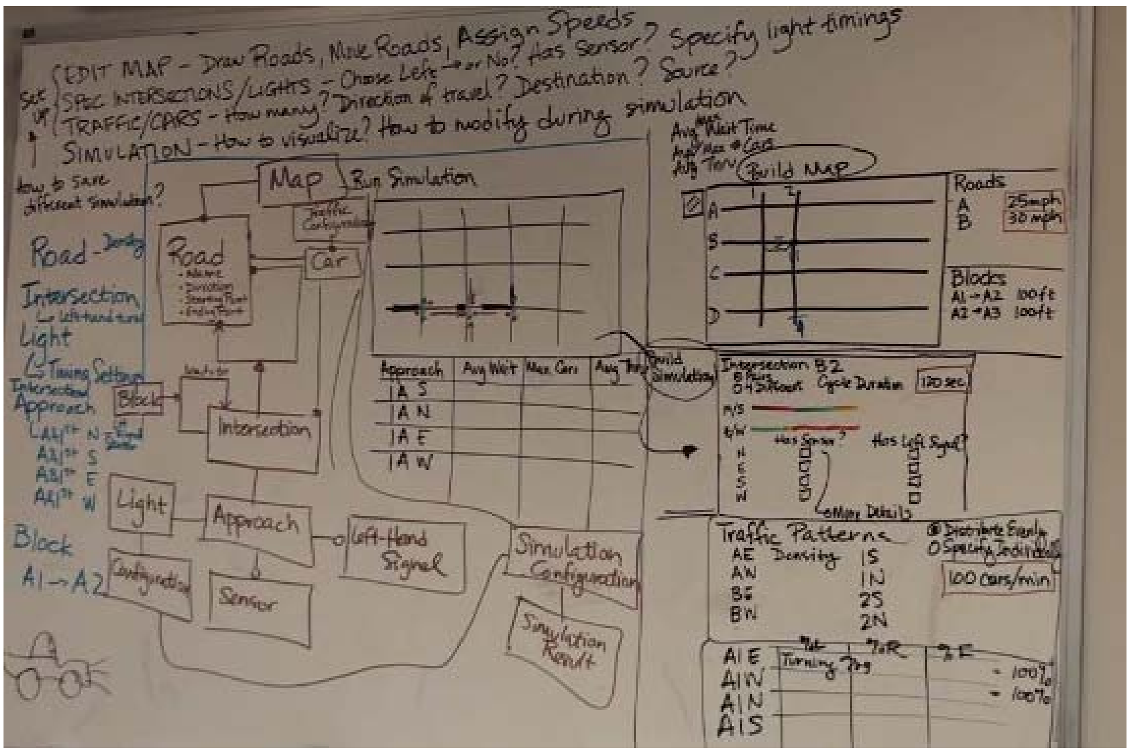

Figure 3. DS2 outcome

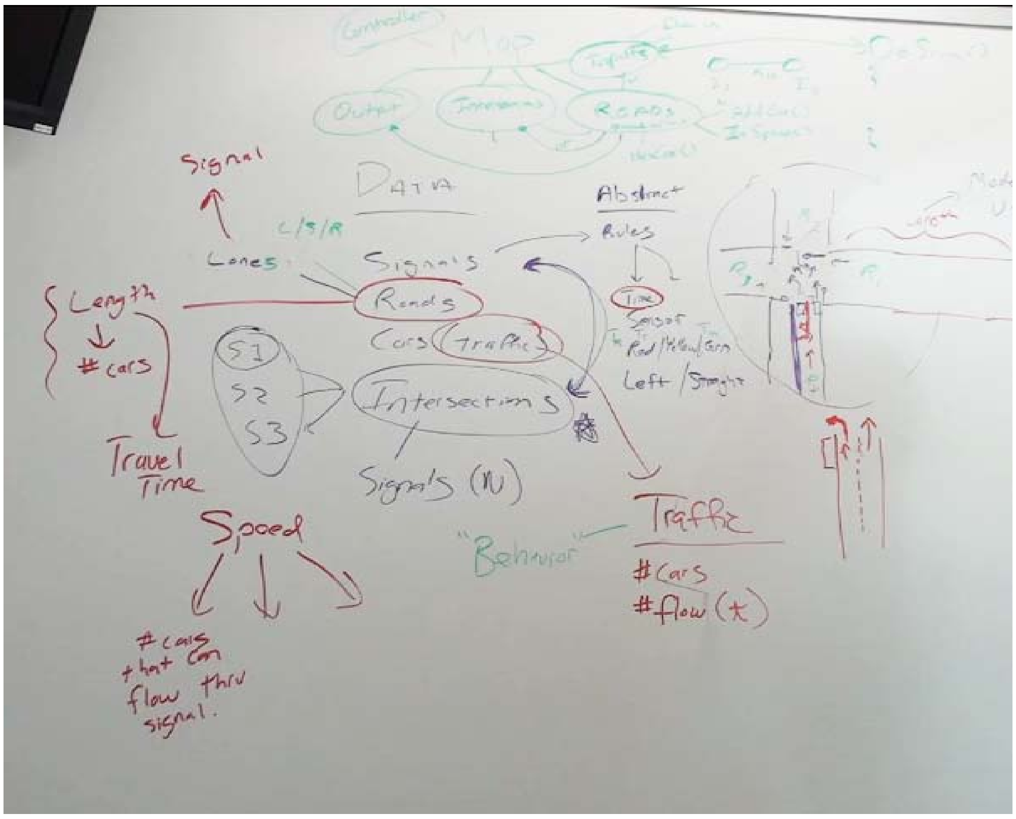

Figure 4. DS3 outcome

Table 3. Diff among discovered concepts

DS1	DS2	DS3
	Approach	
	Block	
Cop		Controller
Car	Car	Car
	Simulation Configuration	
Intersection	Intersection	Intersection
Light	Light	
	Map	Map
Network	Light Configuration	
Queue		
Road	Road	Road
	Sensor	
	Left-Hand Signal	Signal
		Speed
	Traffic Configuration	Inputs
	Simulation Result	Output

B. *Quantitative Analysis Results*

According to the selected metrics, when contrasting DS1 with the union of DS2 and DS3, we can find 7% of innovation (1/15) and around 40% of coverage (6/15). However, when contrasting DS2 with the union of DS1 and DS3, it can be noted a significantly higher innovation, around 33% (4/12), and also a significantly higher coverage, around 75% (9/12). Finally, when contrasting DS3 with the union of DS1 and DS2, we can observe that the innovation level of DS3 (1/15 = 7%) and its coverage level (8/15 = 53%) are both closer to DS1 than to DS2.

Given this analysis, DS2 presented better results in terms of innovation and coverage, if compared to DS1 and DS3. Recalling **Table 1**, we can note that the only design section that adopted prototype as means of expression was DS2. This can lead to an intriguing question: *Do prototype-based design styles lead to higher representational power of the design outcomes, if contrasted to specification-based design styles?* Unfortunately, we cannot answer this question in a general way with the available data, but it represented at least a specific case that this occurred. Additional work should be done in this direction.

While, as mentioned before, we cannot show it effectively (*i.e.,* through rigorous empirical evidence), in the next section we shall try to present some arguments that, in some way, justify the results obtained in this section. In this sense, we show that the prototype-based approach adopted by the DS2 team, although not being totally adherent to the software engineering practices equally named, as explained earlier in this paper, uses the same fundamental idea (*i.e.,* quickly produces means by which users and staff can better understand the system requirements) to more effectively influence the factors that possibly lead to a better design.

However, it is important to note that richest results do not necessarily mean best results. This is because the representational power of the output generated by the design process is intrinsically linked with its abstraction power. Thus, better results are associated to their ability on presenting the relevant points and on hiding not important aspects regarding the problem being solved and the customer involved.

IV. Qualitative Analysis

A first superficial analysis of the provided videos enabled us to observe that none of the teams followed a formal, pre-established process for design. The three design teams were trying to match the requirements while designing, but each team had a different way for doing it. DS1 mainly focused on the system viewpoint. DS2 designed the system while trying to identify the user's needs for that system. Lastly, DS3 was trying to verify whether the design was meeting the system use cases (although the use cases were not explicitly included on their design).

Meanwhile, by carefully analyzing the design sessions, we found that the abstraction procedure adopted by the DS1 and DS3 teams basically looks like the one advocated by the object-oriented paradigm. That is, the design process focuses on mapping real-world entities of the problem into entities in the software-world, called objects [8]. As it can be seen in the videos, the DS1 and DS3 teams extract almost every concept through the analysis (using whiteboard sketching) of one or more everyday situations related to ground traffic.

On the other hand, the DS2 team, besides employing the same methods practiced by the other teams, also adopted something near to what can be called *Human-Computer Interaction* (HCI) design as an important guide during the session. This can be seen by the constant concern of the team on how the system users would interact with it, so that throughout the design session a user interface has been created and adapted to each new understanding of the problem and insight about the solution. Concepts such as Approach and Block came to light primarily by exploiting the interface created on-the-fly by the team.

Therefore, by using a mechanism similar to prototype, each new understanding of the problem and insight about the solution generated a change on the created user interface. By exploiting the interface sketch and usage prediction, sometimes new understandings and insights were created, producing updates in the design of the system and its interface sketch once again.

Although it seems clear that worrying about aspects of the interaction between the system and its users is beneficial to the process of software development, practitioners in the area demands a certain effort to put such concern in practice during software design. As observed by Taylor and Hoek [9], the focus usually considered along software design is primarily restricted to its internal structure and attributes, as for example on how to identify which components and connectors it shall be comprised of. Thus, the way they will expose its characteristics and features is usually disregarded.

A direct consequence of this is a system that does not meet the user's requirements, or when presenting its features they turn out to be inappropriate or even unproductive. Moreover, it can also be argued that it is easier to explain for a programmer why a particular design has been created in such a way using the interactions that the user must make on the system as reference rather than try to explain it by using a background based only in the application domain. For example, it is easier to explain that a particular object must have certain attributes because they are part of the form to be filled in by the user on the web than spending hours explaining that such attributes exist because of some government regulations. And without the support of an interface in the form of a prototype or a sketch it will be difficult to perform the first approach effectively.

Therefore, one possible explanation for a better result in terms of innovation and coverage by the DS2 team can be made based on the methods used during the design session. Such methods, besides involving the usual decomposition of the system using some kind of an object-oriented thought, also had another support by designing the interaction between users and the system. Thus, we can say that the process of abstraction used, in addition to having been influenced by domain factors, was also strongly influenced by HCI factors. The first set of factors represents the dynamics of the problem being addressed in real life. That is, the laws that govern such scenario. In this case, we can cite the laws of physics (*e.g.,* two cars can not occupy the same physical space), traffic laws (*e.g.,* cars must stop at red lights) and consent laws (*e.g.,* who will have preference in crosses). On the other hand, the second group of factors corresponds to the design principles of UI [10, 11], such as maintaining a simple interface, offering the user quick shortcuts to procedures that are apparently more complicated (*e.g.,* intersection settings can be defined with an intuitive click on the intersection of two streets on the map of the system designed by the DS2 team).

As shown in Figure 5, the introduction of HCI design over the traditional software design reveals new factors which could positively influence the creation of appropriate models for it. In the scenario described in this figure, the following problem is proposed: digital scan the entire content of a book that will be used later for learning purposes. Without any concern on how the interaction of the final product with its users will be, *i.e.,* by only observing the domain factors, a product without value would probably be generated in this case, since users of such a system are visually impaired. In this context, the adoption of methods of HCI design over the traditional software design could possibly generate a model more suitable for this application. Note that often it is not just a model with additional features (such as playbacking the entire book), but a totally different way of thinking about the system, thus, potentially encouraging more innovative models and adherent to the user's needs.

Although our analysis primarly focused on the decomposition and abstraction methods used, other aspects (*e.g.,* social and motivational) should be considered in order to gain a better understanding of the analyzed design sessions. In this context, three additional videos were provided with an interview conducted with each pair of designers for a brief overview of the main points of the resulting design. As discussed before, one of the expected results of the sessions is to present the achieved design to a team of software developers who should be able to implement it. During these design summaries, the DS2 and DS3 teams affirmed that they were satisfied with the outcome of their project, considering the time constraint (DS1 team's answer was not included in the related video).

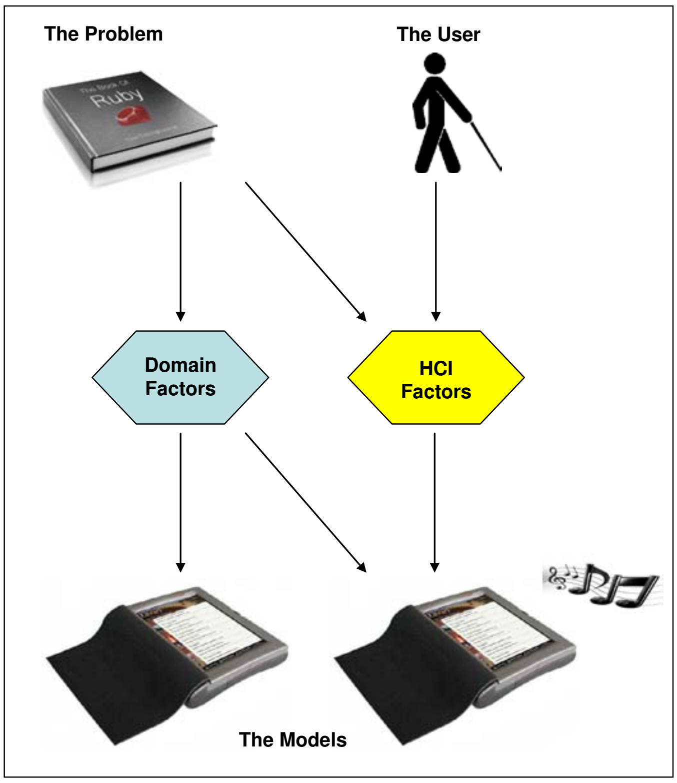

Figure 5. Influence of domain and HCI factors on software models.

During the video analysis, we noted that, although both members were present during DS1 session, most of the time it was conducted by one person – the younger member only made some comments. The team also affirmed that they had never designed together before. DS2 team members seemed to be more integrated; the interaction between them seemed balanced, harmonious and complementary (this is in agreement with the results of the quantitative analysis). Finally, DS3 session was conducted in an ad-hoc manner; it was quite close to an agile design. One of the members mentioned that it is possible to start programming from the session outcome.

We also observed that the academic and professional background of each team somehow contributed to the way the design session was conducted. The DS1 team was used to object-oriented decomposition. The DS2 team worked for years on UI development and interaction design. And the DS3 team has based design in a data-driven process.

V. How Tools Could Help?

Based on the quantitative and qualitative analysis, we present in this section some automation requirements for the design process. We also suggest some tools based on the established requirements.

A. *Automation requirements*

In all sessions, the available space did not seem to be enough for designers to express their thoughts and perceptions. Part of the data written on the whiteboard was deleted (DS1 and DS2) or the unavailable area of the whiteboard was used (DS1). Moreover, some design elements were mixed up with other sketching elements, such as design entities, elements for domain understanding, UI elements, and lower-level domain representations. This indicates that a useful feature for a tool support would be to provide different layers for representing each design view, allowing the identification of relationships and interactions between design elements, and the creation of links for navigating among them. This could lead to traceability in terms of abstraction levels, which can improve comprehension and, consequently, encourage reuse.

Since the members of each team added, deleted, and changed some design parts during the process, another useful functionality for controlling design evolution is fine-grained design versioning. This is especially important for large teams in a distributed environment, considering that concurrent work is a typical scenario in this context. In this kind of tool, a visual diff and merge feature would also be desirable.

Finally, we perceived the need of a mechanism that keeps track of the design rationale. During the design sessions, teams discussed why they were making specific decisions and choosing specific solutions in detriment of other solutions. This knowledge is very important to developers when implementing the design or even to other designers during the design maintenance and evolution. It is important to note that design rationale can be materialized in terms of different kinds of hypermedia contents, such as text, audio (recording the team speeches) or even video (recording the team gestures and reactions).

According to the arguments presented in this section, the following automation requirements should be considered for a tool that aims at supporting software design activities:

- Design layers
- Traceability (among design elements and abstraction levels)
- Navigability (using the previously mentioned traceability)
- Fine-grained versioning
- Visual diff/merge
- Design rationale

B. *Tools suggestions*

We believe that some tools could be used/developed in the context of design, in order to support the teams. We present a high-level analysis of the tools available in software engineering in order to indicate how each tool can support the above requirements.

For solving the need of more space, a sketch-based tool that makes use of open canvas can be used, like [12]. This tool must allow multiple, navigable layers for design, allowing to keep track of different steps or different architectural layers in the same design session.

Regarding traceability, different researches focused on this theme, ranging from the establishment of traceability by construction to the after-the-fact detection of traceability via information retrieval [13], syntactic analysis [14] or data mining techniques [15].

Navigability can be seen as a natural consequence of the traceability adoption. Interesting solutions in this area are found, specially related to the hypertext/hypermedia communities [16,17].

With respect to design evolution, it is interesting to have a version control system for managing changes to design artifacts. As mentioned before, such changes should be considered in a lower level of granularity, so that it can help to identify which changes occurred in a specific element, in a specific context [18,19].

Moreover, since visual strategies (such as the use of different colors, sizes and shapes, or even virtual and augmented reality approaches) can help in understanding the design process and evolution [20,21], it can be interesting to provide UI interaction techniques that can handle cognitive tasks [22].

Finally, design rationale [23] is an important topic in the field of requirements engineering, but its importance goes beyond the requirements phase and permeates all the other phases of the software development process. The existence of design rationale can impose direct influence in the usefulness of the design itself. Thus tools (*e.g.*, [24]) that are able to retrieve or generate this kind of rationale could be particularly important during the software development process. This could be noted when teams were asked to explain what their design intended to express.

In our point of view, as discussed in this section, research is already being conducted regarding all the cited automation requirements. The current challenge is on the integration of these independent and usually conflicting researches into an environment that considers non-functional attributes such as performance and usability. Moreover, an additional challenge is to provide this environment outside a usual desktop computer, but in the context of a ubiquitous design room. This scenario would provide greater flexibility to designers, allowing them to freely express their creativity, without incurring in the problems discussed throughout this paper.

VI. CONCLUSION

In this paper we analyzed through video sessions how three professional teams performed design. This analysis occurred in terms of design styles and outcomes, and first consisted on the identification of the design style of each team and the produced outcomes. After that, we performed a quantitative and qualitative study, comparing the completeness of the design outcomes to infer their representational power. Finally, based on our analysis, we established some requirements for tools builders that would help on diminishing the design flaws perceived in the video sessions.

In our point of view, the main contribution of this paper is the establishment of an initial relationship among the influence of design style on the representational power of the design outcomes. However, additional contributions, such as a fair and independent analysis of professional designers in action and the suggestion of automation requirements for building design tools, are also relevant and may foment future work.

Through our qualitative analysis of the results, the paper also reinforced the intuitive but few practiced idea that the design process of the internal structure of the software should be as important as the process of HCI design, so that both activities appear to be complementary towards the development of software with quality. That is, software that fully meets the requirements imposed by the problem being addressed, called throughout the paper as domain factors, and the users' requirements and their needs for interaction.

In this context, one important question remains open for research in software design: *how to effectively combine into a single design discipline techniques and tools widely used in traditional object-oriented design with tools and techniques used in HCI design?* We see that the synergy of both practices in a common development framework could lead to more representative outcomes, which in turn could lead to more innovative and effective products in their application areas.

However, it is important to emphasize that design processes are usually driven by people with different academic and professional background, using different methodologies (each one with a particular level of detail) that can produce several kinds of outcomes. We did not intend to assess which design session was "the best one". Additionally, rather than one single, time-limited session, a typical software design process can take days or even months. Thus, our analysis cannot be generalized, but our hypothesis stated above can point out a research direction for those who want to get involved with the subject in discussion.

Future works include (i) experiments for examining how different programmers would code each design outcome; this could provide feedback on whether the design outcome was suitable and sufficient, based on the divergences between design and code, and how much and which additional information is needed; (ii) executing a design inspection, checking for imprecision, ambiguity, mistakes and so on; and (iii) contrast professional software design and academic software design, considering the same requirements set.

ACKNOWLEDGMENTS

Our thanks to the companies that provided their design session videos, and to CAPES and CNPq for their financial support.

REFERENCES

[1] T. Winograd, *Bringing Design to Software*, ACM Press, 1996.

[2] J. Kramer, "Is abstraction the key to computing?," *Commun. ACM*, vol. 50, 2007, pp. 36-42.

[3] M. Shaw, "Comparing architectural design styles," *Software, IEEE*, vol. 12, 1995, pp. 27-41.

[4] L. Mathiassen e J. Stage, "The principle of limited reduction in software design," *Information Technology & People*, vol. 6, 1990, pp. 171 - 185.

[5] E. Gamma, R. Helm, R. Johnson, e J.M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.

[6] J. Frederick P. Brooks, "No Silver Bullet Essence and Accidents of Software Engineering," *Computer*, vol. 20, 1987, pp. 10-19.

[7] R. Baeza-Yates e B. Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley, 1999.

[8] G. Booch, "Object-oriented design," *Ada Lett.*, vol. I, 1982, pp. 64-76.

[9] R.N. Taylor e A.V.D. Hoek, "Software Design and Architecture The once and future focus of software engineering," *2007 Future of Software Engineering*, IEEE Computer Society, 2007, pp. 226-243.

[10] D.A. Norman e S.W. Draper, *User Centered System Design: New Perspectives on Human-computer Interaction*, CRC, 1986.

[11] B. Myers, "Challenges of HCI design and implementation," *interactions*, vol. 1, 1994, pp. 73-83.

[12] N. Mangano, A. Baker, e A.V.D. Hoek, "Calico: a prototype sketching tool for modeling in early design," *Proceedings of the 2008 international workshop on Models in software engineering*, Leipzig, Germany: ACM, 2008, pp. 63-68.

[13] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia, e E. Merlo, "Recovering Traceability Links between Code and Documentation," *IEEE Trans. Softw. Eng.*, vol. 28, 2002, pp. 970-983.

[14] L.C. Briand, Y. Labiche, e L. O'Sullivan, "Impact Analysis and Change Management of UML Models," *Proceedings of the International Conference on Software Maintenance*, IEEE Computer Society, 2003, p. 256.

[15] A.T.T. Ying, G.C. Murphy, R. Ng, e M.C. Chu-Carroll, "Predicting Source Code Changes by Mining Change History," *IEEE Trans. Softw. Eng.*, vol. 30, 2004, pp. 574-586.

[16] K.M. Anderson, R.N. Taylor, e J. E. James Whitehead, "Chimera: hypertext for heterogeneous software environments," *Proceedings of the 1994 ACM European conference on Hypermedia technology*, Edinburgh, Scotland: ACM, 1994, pp. 94-107.

[17] T.N. Nguyen, E.V. Munson, e J.T. Boyland, "The molhado hypertext versioning system," *Proceedings of*

the fifteenth ACM conference on Hypertext and hypermedia, Santa Cruz, CA, USA: ACM, 2004, pp. 185-194.

[18] L. Murta, H. Oliveira, C. Dantas, L.G. Lopes, e C. Werner, "Odyssey-SCM: An integrated software configuration management infrastructure for UML models," *Science of Computer Programming*, vol. 65, Abr. 2007, pp. 249-274.

[19] D. Ohst, "A Fine-Grained Version and Confguration Model in Analysis and Design," *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, IEEE Computer Society, 2002, pp. 521-527.

[20] S. Wenzel, J. Koch, U. Kelter, e A. Kolb, "Evolution analysis with animated and 3D-visualizations," *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, 2009, pp. 475-478.

[21] S. Eick, T. Graves, A. Karr, A. Mockus, e P. Schuster, "Visualizing software changes," *Software Engineering, IEEE Transactions on*, vol. 28, 2002, pp. 396-412.

[22] B.E. John e D.E. Kieras, "Using GOMS for user interface design and evaluation: which technique?," *ACM Trans. Comput.-Hum. Interact.*, vol. 3, 1996, pp. 287-319.

[23] J. Lee e K. Lai, "What's in design rationale?," *Human-Computer Interaction*, vol. 6, 1991, pp. 251-280.

[24] C. Dantas, L.G.P. Murta, e C.M.L. Werner, "Mining Change Traces from Versioned UML Repositories," *Proceedings of the Brazilian Symposium on Software Engineering (SBES'07)*, 2007, pp. 236-252.