Software engineering issues in interactive installation art

Anna Trifonova* and Letizia Jaccheri

Department of Computer and Information Science,
Norwegian University of Science and Technology,
Sem Saelands vei 7-9,
7491 Trondheim, Norway
Fax: +47 7359 4466
E-mail: trifonova@idi.ntnu.no
E-mail: letizia@idi.ntnu.no
*Corresponding author

Kristin Bergaust

Faculty of Art, Design and Drama,
Oslo University College,
Pilestredet 46,
0130 Oslo, Norway
E-mail: kristin.bergaust@est.hio.no

Abstract: Software engineering has been in contact with new media art for years, although the connections between these fields have rarely been explicit. In this article, we discuss the software engineering issues that appear in one of the new media art subfields, namely interactive installation art, as reported in the scientific literature. Interactive installations are often built in small multidisciplinary teams, including collaboration between artists and software developers, as the artworks are heavily dependent on software. Some of the requirements for successful multidisciplinary work are to share common goals and common language and thus software engineering issues in art context are important to consider and discuss. The goal of this article is twofold. On one hand we provide a road map for artists to the basic software engineering concepts and terms for the purpose of communicating with software engineers or for acting themselves as programmers of their artworks. On the other hand, software engineers who start working with interactive installation art will profit from this summary of relevant reports. Awareness of the software engineering essentials and knowledge of others' experience in similar situations would allow good planning, recognition of problems in early stages, and might simplify the look up for solutions in the right direction.

Keywords: development of interactive installations; interactive installation art; new media art; software engineering.

Reference to this paper should be made as follows: Trifonova, A., Jaccheri, L. and Bergaust, K. (2008) 'Software engineering issues in interactive installation art', *Int. J. Arts and Technology*, Vol. 1, No. 1, pp.43–65.

Biographical notes: Letizia Jaccheri is a Professor at the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU) since 2002. In the last five years, she has been involved in

Copyright © 2008 Inderscience Enterprises Ltd.

the supervision of at least 10 PhD students. She has more than 15 years experience with research projects, both at National level (Italian and Norwegian) and international. she has been working with software research issues since 1988. She wrote her PhD on software process modelling in 1994. She is interested in software intensive processes with special focus on artistic software and open source software.

Anna Trifonova has graduated at New Bulgarian University (Sofia, Bulgaria) in 1999 with specialty 'nformation Systems and Technologies - applications in business and office' In March 2006 she finished her PhD at the International Graduate School of Information and Communication Technologies at the University of Trento (Italy). Her research topic was 'Mobile Learning: Wireless and obile Technologies in Education'. Since January 2007, she has a post-doc position at NTNU with a research grant from ERCIM - the European Research Consortium for Informatics and Mathematics. Her research interests in software and art also extend to their intersection with mobile computing and education.

Kristin Bergaust has graduated from the University of Oslo, Norway in 1983 and The National Academy of Fine Art in Oslo 1991. She was the director of Atelier Nord medialab for artists 1997–2001 and an active artist, curator and organiser in the field of new media, technology and art in Norway and elsewhere in Europe. Since 2001, she has been a Professor at Trondheim Academy of Fine Art, Faculty of Architecture and Fine Art, NTNU Norwegian University of Technology and Science. She has recently accepted a position as a Professor at the master programme, The Faculty of Art, Design and Drama of Oslo University College. Her artistic interests includes cultural change and exchange, communication technology and flaws of communication. Ongoing projects relate to Latin–American immigrant cultures and use position of Web 2.0 applications in artistic contexts.

1 Introduction

Computer art dates back to the 1960s. The first computer art exhibition took place at Technische Hochschule in Stuttgart in 1965. The same year at the Howard Wise Gallery in New York city, the earliest computer art exhibition took place in the USA (Digital Art Museum, 2007).

The first software engineering conference was held in Garmish in 1968 (Naur and Randell, 1968). Software engineering and computer art, blooming at the same time, have met several times, even if these relationships may not have been rendered explicit.

The use of digital technology in contemporary art is often referred to as new media art. Since, the early 1990s within the new media art realm there is a growing production of interactive art installations.[1] These artworks are generally complex and they are heavily dependent on software for controlling the whole system. The production of the software might be fully or partially done by the artist who takes the role of software developer. In many cases, however, the needed software is rather complex, involving the use of numerous programming languages, tools and software technologies and thus, the need for involvement of programmers and software engineers.

As in other countries, interactive installations are often made in Norway – several examples might be seen in Lysaa, Sandboe and Kvinnsland (2006). Within the Software Engineering group of the Norwegian University of Science and Technology (NTNU) we have started a project, called SArt (http://prosjekt.idi.ntnu.no/sart/). The members of SArt participate in multidisciplinary projects for the development of interactive installations that involve collaboration between software engineers and artists. These projects motivate us for a profound investigation in this domain and our goal is to find, expose and bridge possible gaps between the two fields.

This article contains several contributions. On one hand, artists will find useful the list of software engineering issues with their descriptions. Artists who are interested and experienced in new media art field are not totally foreign to scientific discourses and are generally interested in how they can communicate with people who can assist their projects. Here, we present in a simple but structured way the main software engineering concepts. Afterwards, we put these concepts within the context of interactive installation art. This knowledge will help artists work in projects with software intensive systems. Building a common language between artists and software engineers is one of the advantages. Meanwhile, artists who work on their own, taking the role of software developer, might profit from indications of where problems might appear and what solutions might be appropriate, thus improve planning.

On the other hand, software engineers in interactive installation art projects will profit from practitioners' experience reported in the literature and summarised here. This includes a list of utilised tools. We also discuss some directions in software engineering which require further research and even shift in the utilised approaches, as they might need to adapt to the specifics of the artistic processes.

The literature review performed and presented in this article (Section 5), covers the scientific publications which we found as part of a systematic review (Trifonova, Ahmed and Jaccheri, 2007). It reveals practical problems, technical solutions and software engineering issues in interactive installation art. Although in many cases, one or more of the authors of the cited articles are artists literature review shows to be an inadequate methodology for discovering artists' viewpoint and experiences on these issues.

Further, this article is organised as follows: in Section 2, we shortly discuss interactive installation art field and its specifics, namely, multidisciplinarity and interactivity. In Section 3, we synthesise the major software engineering concepts and show in Section 4 how they apply to the production of interactive installations. Section 5 presents a summary of the software engineering issues in installation art projects and shows different approaches found in the literature. A discussion in Section 6 and conclusions in Section 7 are followed by references.

2 New media art and interactive installation art

New media art is an area within contemporary art that involves the use of new media technology. In this article, we do not try to give an exact definition of new media art, we do not limit what it includes and we do not intend to define how it relates to other art domain. Instead, we point to the work of new media art theoreticians such as Manovich (2002a) and Tribe, Jana and Grosenick (2006) and reference the practitioners. Biswas and Singh (2006) define new media art as

> "type of new media application where an artistic idea is expressed using technology or new media artifacts."

As software engineers, we have to be critical about the notion of new media as what used to be new in the early 1990s, for example, the HTML language and web browsers, is now main stream technology. Web 2.0 which can be regarded as new at the time of writing will not be new in a couple of years from now. Manovich gives an explanation of what artists intend as new in the new media art:

> "new media today can be understood as the mix between older cultural conventions for data representation, access and manipulation and newer conventions of data representation, access and manipulation. The 'old' data are representations of visual reality and human experience, i.e. images, text-based and audio-visual narratives – what we normally understand by 'culture'. The 'new' data is numerical data." (Manovich, 2002b)

Installation art is a phenomenon which might be traced back to artists like Marcel Duchamp - artwork Fountain, 1917. Interactive installations are one development within installation art and also part of new media art, because of "their origins in, and reliance upon, computer-based technology" (Steinkamp, 2001). An interactive installation includes a physical construction which is often placed in a public space. Usually, certain elements of the installation are changing in time (e.g. video, audio, mechanical parts movement, etc.). Often, these changes are due to spectator(s) presence and/or actions. The creation of an interactive installation commonly requires specialists with different areas of competence to collaborate with the artist.[2] A multidisciplinary team can involve artists, art curators, constructors, hardware designers, architects, electrical engineers, software engineers, programmers, etc.

Interactivity is a major issue in interactive installations. Different interactivity types might be defined. In fact, interaction is extensively discussed in (almost) all articles describing interactive installations.

In Hannington and Reed (2002), three distinguished types of interaction in multimedia applications are discussed: *passive* interaction is where the content has a linear presentation and users interact by only starting and stopping the presentation; *interactive* is when users are allowed to choose a personal path through the content; *adaptive* is the interaction in which users are able to "enter their own content and control how it is used".

In Sommerer and Mignonneau (1999), the authors discuss two types of interaction that they have observed in existing interactive artworks: *pre-designed* or *pre-programmed* paths of interaction, as in interactive CDs where the viewer can choose his/her path, but the possibilities are limited; and *evolutionary*[3] interaction in which the artwork's processes are linked to interaction and is evolving continuously.

Edmonds, Turner and Candy (2004) discuss four categories of "relationship between the artwork, artist, viewer and environment": *static, dynamic-passive, dynamic-interactive* and *dynamic-interactive (varying)*. While the first is a lack of interaction, the latter three describe situations in which the artwork responds to its context. In dynamic-passive, the artwork response is triggered by environmental factors as temperature, humidity, etc. In dynamic-interactive, in addition to the environmental factor the human presence and/or actions (purposeful or not) are captured and are used as parameters for changing the artwork. The rules about how the parameters are treated are static in this case. When an agent (either human or programme) is modifying its original specifications the artwork is dynamic-interactive (varying).

We find not only these three categorisations of interaction important, but also each one is incomplete. In our viewpoint, there are three perspectives to be taken into consideration – what content is shown to the audience (i.e. pre-defined by the artist, algorithmically generated or inputted by the user); what triggers interaction (audience presence/actions or environmental parameters) and how the artwork or its content is influenced by the surroundings and the audience (what interaction rules are applied). Using these three properties, we can cover all above described interaction types (see Table 1).

Table 1 Interactions in interactive installation art

		Interaction rules		*Triggering parameters*			*Content origin*		
		Static	*Dynamic*	*Human presence*	*Human actions*	*Nature/ Environmen*	*Predefined by the artist*	*Users Input*	*Generated/ Algorithmic*
Hannington and Reed (2002)	Passive				x		X		
	Interactive	X			X		X		
	Adaptive		X		X			X	
Sommerer and Mignonneau (1999)	Pre-designed	X		x	X		X		
	Evolutionary	x	x	x	X				X
Edmonds, Turner and Candy (2004)	Static						X		
	Dynamic-passive	X				X	X		
	Dynamic-interactive	X		X	X		X		
	Dynamic-interactive (varying)		X	X	X		X		

Note: the small ‘x’ indicates that the cited author did not clearly state the inclusion of the marked option in their category, but we see it as mutually included.

1 *Interaction rules*. The rules that control the interaction might be static or dynamic. In Edmonds, Turner and Candy (2004), this difference is shown by introducing the *dynamic-interactive (varying)* category. The *evolutionary* interaction in Sommerer and Mignonneau (1999) is also based on dynamic-interaction rules, but limits the rules to evolutionary algorithms.

2 *Triggering parameters*. Generally, the interaction rules depend on environment parameters that are changed by the audience. Triggering parameters might be captured by wide variety of sensors: sound, light, movement, temperature, humidity pressure, etc. They might be also other type of data input, like the date, the moon phases. Most often the audience is directly participating in the interaction intentionally, but it is possible that no intention is required and only the spectators’ presence is enough to trigger the interaction rules. However, in some cases the changes in the artwork might not depend on the audience at all, but only on the environment – such option is foreseen only in Edmonds, Turner and Candy (2004) with the *dynamic-passive* interaction category.

3 *Content origin*. Whether the artwork presents visual or audio content to the spectators this content might be dynamically generated or pre-defined by the artist. The pre-defined content might also be dynamically manipulated. In particular cases, the audience might also input content to the artwork, for example, by sending

pictures/music from their phones. Such option is only seen in Hannington and Reed (2002) – category *adaptive*.

3 Software engineering

Here, we make an attempt to synthesise software engineering theories (Bourque et al., 2004; Pressman, 2004; Conradi, 2007) for the purpose of defining the intersection between software engineering and interactive installation art in the following sections of the article. In other words, this list points to software engineering terms and concepts that one can use to reflect about software development and maintenance in new media art. We have developed this list of concepts by looking at the topics of the latest International Conference of Software Engineering (ICSE 2007, www.cs.ucl.ac.uk/icse07), and we have modified it supported by our experience and discussions with colleagues. These concepts are:

1 *Requirements*. Software requirements are the real-world goals, needed functionality and constraints for the software to be developed. The process of software requirements engineering includes identifying the stakeholders and their needs and documenting these for analysis and implementation. For a software development project to be successful, the software engineers and the client have to agree on the requirements to be implemented. Software requirements might be functional and non-functional. Functional requirements define the functionalities that the finished software will offer to the end-user. Non-functional requirements define the qualities required for the system such as execution speed, stability, etc. (see point 8 – quality attributes). More about requirements in Wiegers (2003).

2 *Software architecture and design*. Software architecture is a description of the high-level design of a system (i.e. the product), its main parts and their relations and interactions. Software design is the process of making and analysing such architectures. Details may be found in Bass, Clements and Kazman (2003).

3 *Evaluation, validation and testing*. Validation (of both process and product) means showing that a delivered product/system satisfies the user's real or future needs. Testing is the controlled execution of programme code in order to find and correct errors (Østerlie and Wang, 2007). There are different levels of testing (e.g. unit, module, subsystem, system, acceptance, etc.) to check that actual execution with given inputs produces the expected results.

4 *Process models and project management*. Process models describe the activities with ordering and compositional relations, artefacts being produced or consumed by such activities, human work roles, what tools/techniques to use, and possibly what measurements to apply (i.e. a formalisation of the software development process – e.g. agile). Example of a traditional highly used process model is the 'waterfall' model where the process generally flows consecutively through the phases of requirements elicitation, design, implementation, testing, integration and maintenance. Another example is the agile software development (Cockburn, 2006), introduced more recently. It is based on the principles of iterative development (short cycles lasting a couple of weeks) where each iteration follows the full software development life-cycle (i.e. requirements elicitation, design, implementation, testing). Agile methods, in contrast to more traditional ones, emphasise on

face-to-face communication rather than on software documentation, on close collaboration with the customer rather than on requirements agreements, etc. adaptive software development, mentioned further in the article, is one of the agile models. Project management deals with planning and control of execution process model, including schedulers, budget, etc. (Project Management Institute, 2004).

5 Development environments and tools (e.g. Eclipse, www.eclipse.org) – are programmes used by software engineers and programmers as aid for the software design and implementation, synonyms are Computer Aided Software Engineering (CASE) or integrated process support environment.

6 *Maintenance (software maintenance)*. Further development of a software product after its first release, also usually organised as a project. Two-third of total software costs may fall on software maintenance. We distinguish between perfective (new or revised requirements), adaptive (new technologies/platform), corrective (fixing faults) and preventive maintenance (internal reorganisation). Reuse is a way of software development that includes systematic activities for creation and later incorporation ('reuse') of common, domain-specific artefacts. Reuse may have profound technological, practical, economic and legal obstacles – but the benefits may be huge. It mostly concerns programme artefacts in the form of components. Standard use of platform components – i.e. commodities such as operating systems, data base management systems, internet netware, graphical user interface, etc. – are normally not called reuse. Read more about maintainability in Østerlie and Wang (2006).

7 *Open source software*. Software for which the access to the source code is open, the distribution and re-distribution is global, free, the licenses are non-restrictive, etc. Traditionally, open source software has been developed by interested communities and software users and developers have been volunteers. In the recent years, large and small enterprises (among the large ones we mention IBM and Sun Microsystems) are exploiting business model centered around production, customisation and service of OSS products (Østerlie and Jaccheri, 2007).

8 *Quality attributes (performance, reliability, portability, reusability, modifiability, scalability, security, safety, etc.)*. Performance is the measuring of the speed or volume offered by a service, e.g. delay/transmission time for data communication, storage capacity in a database, image resolution on a screen or sound quality over a telephone line. Reliability is the probability of failure-free behaviour (vs. stated requirements), in a specific context (executing environment and usage profile) and time period. Portability refers to the possibility and ease to move (components of) a software system from one environment to another. Reusability measures indicate the possibilities to reuse components between systems. Modifiability concerns the simplicity to make changes in the system in several levels (i.e. local, non-local and architectural). Scalability specifies the ability of a software system to handle grouth. Security is the level of protection against unauthorised access (e.g. read/write/search) of data/information. Safety is the degree of protection against dangerous events, i.e. events with possible serious consequences for humans, environment, business and/or society. Quality attributes are defined during requirements elicitation (Wiegers, 2003).

4 Interactive installations as software engineering products

In 2005, Oates (2006) proposes to extend Information System (IS) research in the domain of computer art. The first suggestion is that "computer art might be seen as a kind of information system". In this section, we extend this idea and discuss more concretely the mapping between an interactive installation and a software engineering product. Schematically, a software engineering product might be presented as a black box that receives a digital input, processes it and the result is outputted to the user. Interactive installations might be mapped to this schema, as they receive certain input that is digitally processed and the output is given back to the audience (Figure 1).

Figure 1 Interactive installation as information system (see online version for colours)

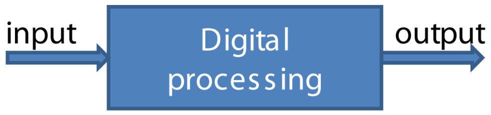
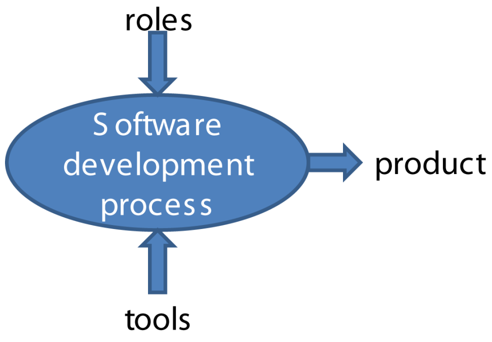


Figure 2 Software development (see online version for colours)



Machin (2002) states that the technology for controlling the interactive installation artwork in general does not differ from the technology used for controlling industrial machines; what differs is that the artists require the technology to be more accessible, so that they can experiment while creating the artwork.

We would like to extend this idea by adding the details around – by describing the processes involved, the stakeholders and their roles and the tools used.

4.1 Product

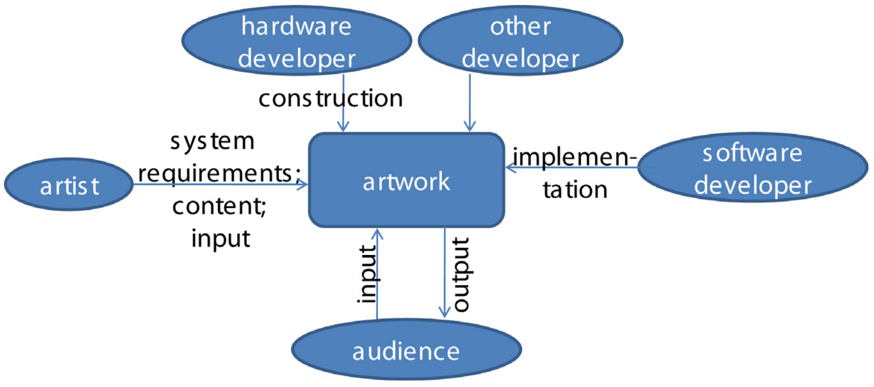
The final product in the creation of an interactive art installation is the artwork as a whole. This includes its hardware (i.e. the physical construction or sculptural elements of the artwork) which is physically placed as desired by the artist in its context (e.g. public spaces, galleries, etc.). It also includes its software (i.e. programmes) and content (e.g. pre-produced visual or audio material).

However, in software engineering terms the product is the software that controls the interactive installation. The software system takes care of the input (e.g. data from motion detectors, light sensors, images from video cameras, content sent by the audience or pre-defined by the artist, etc.), applies certain digital processing (for example applies the pre-defined interaction rules) and gives the output to the audience (e.g. plays generated audio, shows adapted video/animation or gives the appropriate signals to move certain parts of the construction). Additional products, such as documentation, user manuals, website, supplementary software tools, etc. might also be expected.

4.2 Roles (stakeholders)

A project that aims to develop an interactive art installation is often multidisciplinary, as it involves production of physical and software components. The stakeholders of such installation might include one or more of the following: not only the artist, the software designer and developer, the hardware designer and developer, etc., but also the final audience/spectators (Figure 3).

Figure 3 Interactive installation art stakeholders (see online version for colours)



The artist has the key role in the project. He/she comes with the idea of the whole system. The artist might have a global view of what message the artwork should send to the audience or what reaction it should trigger. It is possible, however, that the artist's goal is to experiment with certain technology without having defined in advance an aim or message for the audience. The artist might be seen as a client to the software engineering team. He/she has to agree on the software requirements, the final product quality attributes, projects scheduler, etc. In addition, behind the artist there might be sponsors or commissioners who might be setting the budget and the schedule constraints.

Software and hardware engineers have the mission to convert the artist's desires and visions about the artwork into formal requirements that are later implemented into the final product within the time and the budget available. Depending on the requirements and the technology involved software engineers propose appropriate process model, software architecture and tools for implementation.

The audience/spectators participate at the final stage, when the whole system is ready, integrated and put in place. The audience influences the artwork by its presence or by its actions and in this way the spectators are becoming part of art.

The roles which we represent here might be blurred and overlapping in the real-world. Often, the artist is also a programmer/software developer. It might be that the same person takes diverse roles in two distinguished projects – 'artist' in one project and 'programmer' in another.

4.3 Tools

As mentioned in Section 2 (point 5), software engineering tools refer to development environments – software that aids in the design and the implementation of the final product or in the management of the software project. Many different CASE tools are available commercially or free of charge and practitioners choose the most suitable ones for their task, thus different tools might be used by people with different roles in a project.

In some cases, the final product of a project might be a tool. For example, the software development team might implement a software system that the artist will use for experimenting with the artwork design or for implementing and controlling the interaction.

5 Software engineering issues in interactive installation art

In this section, we link the concepts discussed in Section 3 to academic papers which discuss interactive installations. We show a synthesis of 14 papers describing 25 interactive installations (see Table 2).

Table 2 Interactive installations found in our literature review

Installation	*Exhibited and supported (where, when, by whom)*	*Artists/Scientist*[a]
Locative sound-scape (Biswas and Singh, 2006)	Exhibited at Park Emile Gamelin, Montreal Supported by Concordia University; Hexagram: Institute for Research/Creation in Media Arts and Technology, etc.	Scientists: Amitava Biswas and Jagmit Singh
Fifteen seconds of fame (Solina, 2004)	Exhibited several times, first at the 8th International Festival of Computer Arts, 28 May–1 June 2002, Maribor, Slovenia	Franc Solina – artist and scientist and four Master Students in CS, Slovenia
game_of_life (Satomi and Sommerer, 2007)	–	M. Satomi and C. Sommerer
A-Volve (Sommerer and Mignonneau, 1999)	Multiple exhibitions 1994–2007, first at Ars Electronica '94 Supported by ICC-NTT Japan, and NCSA, Urbana and IL, USA	Artist and scientist: Christa Sommerer, Laurent Mignonneau and Tom Ray (A-Life scientist)
Books of sand (Sardon, 2006)	Several exhibitions, including Daum Museum of Contemporary Art, Missouri, USA and Museum of Modern Art of Buenos Aires. MAMbA (2002–2003) Supported by UCLA Theater, Film and TV Department (residency programme), Univeristy of Tres de Febrero and Linda Lighton Foundation of Kansas City	Artist: Mariano Sardón Scientist: Laurence Bender
Gender specific (Steinkamp, 2001)	Exhibited at Santa Monica Museum of Art, and Bliss House, Pasadena, California. November 11– 18, 1989 (simultaneous one-person exhibitions) and Part of the LA Freewaves video festival. Supported by Foundation for Art Resources	Artist: Jennifer Steinkamp

Table 2 Interactive installations found in our literature review (continued)

Installation	*Exhibited and supported (where, when, by whom)*	*Artists/Scientist*[a]
Genetic Manipulator (GENMA; Sommerer and Mignonneau, 1999)	Exhibited at Ars Electronica Center (AEC) in Linz, Austria, as part of a permanent exhibition, 1996; extended several times Supported by ATR Media Integration and Communications Research Lab, Kyoto, Japan	Artist and Scientist: Christa Sommerer Laurent Mignonneau
Iamascope (Edmonds, Turner and Candy, 2004)	Exhibited several times since 1998, including Play Zone, Millenium Dome 2000, Video Construct, Kyoto Two	Artist and Scientist: Sidney Fells
Interactive plant growing (Sommerer and Mignonneau, 1999)	Permanent collection of the Media Museum at the ZKM Karlsruhe, Germany, 1997	Artist and Scientist: Christa Sommerer and Laurent Mignonneau
Intro Act (Sommerer and Mignonneau, 1999)	Exhibited at Biennale de Lyon at the Musée d'Art Contemporain in Lyon, France, as part of the museum's collection, 1996/1997	Artist: Christa Sommerer, Artist and Scientist: Laurent Mignonneau
Memichi (Biswas and Singh, 2006)	Exhibited at Banff national park, Alberta, Canada	Scientist: A. Biswas
MIC Exploration Space (Sommerer and Mignonneau, 1999)	Exhibited at ATR Media Integration and Communication Systems Laboratories in Kyoto, Japan, 1996	Artist and Scientist: Christa Sommerer and Laurent Mignonneau
Nautilus (Strömberg, Väätänen and Räty, 2002)	Supported by VTT Information Technology, Cube Oy, Nokia Research Center, Särkänniemi Adventure Park, Tekes, the National Technology Agency and the University of Lapland, Finland	Scientist: Hanna Strömberg, Antti Väätänen and Veli-Pekka Räty
Phototropy (Sommerer and Mignonneau, 1999)	Exhibited in Shiroishi, Japan 1998 Supported by Artifices, Saint-Denis, France	Artist: Christa Sommerer, Artist and Scientist: Laurent Mignonneau
Priva-Lite Panel Construction "Digital Garden" (Machin, 2002)	Exhibited outdoors in Loughborough, UK, 1998 Supported by COSTART project and Gallery of the Future, Loughborough, UK	Artist: Esther Rolinson Scientist: Colin Machin
Cycles (Bestor, 1996)	–	Artists: B. Cornett, J. Wade and C. Bestor
Remote furniture (Fujimura, 2004)	Exhibitions in Yokohama, Queens mall, Aug 1999; Tokyo, Ginza subway station, August 1999	Artist: Fujimura, Noriyuki
Stiffs (Steinkamp, 2001)	Art Center College of Design's Williamson Gallery, 2000 Supported by ACME Gallery, Los Angeles	Artist: Jennifer Steinkamp in collaboration with Jimmy Johnson (soundtrack)
SwarmArt Boyd, Hushlak and Jacob (2004)	Two installations for a Calgary gallery, 2002–2005	Artist: Gerald Hushlak Scientist: Jeffrey E. Boyd and Christian J. Jacob
SWELL (Steinkamp, 2001)	Several exhibitions until 2005, first ACME, Santa Monica, California. (One-person exhibition) November 10 – December 8, 1995 Supported by The Museum of Contemporary Art, Los Angeles, California with funds provided by the Ruth and Jake Bloom Young Artist Fund	Artist: Jennifer Steinkamp in collaboration with Bryan Brown (soundtrack)

Table 2 Interactive installations found in our literature review (continued)

Installation	*Exhibited and supported (where, when, by whom)*	*Artists/Scientist*[a]
Swimming across the Pacific (Fels et al., 2005)	–	Artist: Alzek Misheff, Fels S. Scientist: Fels S., Kinoshita Y., etc.
The TV room (Steinkamp, 2001)	Santa Monica Museum of Art, 1998 Supported by ACME., Los Angeles	Jennifer Steinkamp in collaboration with Andrew Bucksbarg (soundtrack)
Trans Plant (Sommerer and Mignonneau, 1999)	Permanent collection of the Tokyo Metropolitan Museum of Photography, Tokyo, Japan, 1995–1998	Artist and Scientist: Christa Sommerer and Laurent Mignonneau
Trigger (Marchese, 2006)	Supported by Advanced Telecommunications Research (ATR) Laboratories, Japan. Pace University Digital Gallery, New York, NY, October 18 – November 8, 2005	Artist: Jody Zellen scientists from Pace University's Center for Advanced Media
Untitled (Steinkamp, 2001)	Several exhibitions between 1993 and 2006, first at FOOD HOUSE, Santa Monica, California, '93	Jennifer Steinkamp

[a] By scientist in table, we refer to Software Engineers/Developers.

These articles were found within a larger study – a systematic literature review on the intersection between software engineering and art. With some exceptions (i.e. some articles were pointed to us by colleagues) the articles were found by searching IEEE Xplorer, ACM Digital Library, Google and the NTNU library meta-search engine (www.bibsys.no) with a combination of the keywords 'art installation', 'software engineering', 'artist and software'. More information about the complete literature review process and initial results is available in Trifonova, Ahmed and Jaccheri (2007).

5.1 Requirements

During the development of a software product, requirements definition is a task of major importance. It starts at an early stage of the project and might continue to evolve, update, increase or change throughout the near end of the project. Highly appreciated by software engineers are projects where the agreement on software requirements is reached with the client at the initial stage and they are locked against changes.

Changes in requirements, especially in later stages, might lead to drastic changes in the software architecture design and implementation and thus, might cause increase in project cost and delays in the scheduler.

As in many other fields, requirements definition is one of the most difficult tasks for the software developers in the interactive installation projects – "It is the most important part of the process because without a precise understanding of the system requirements it is possible to build a well functioning system that does not perform the tasks requested by the user" (Marchese, 2006). However, requirements are often found difficult to capture. (Machin, 2002) underlines that requirements definition is the hardest part and states that "we find the greatest challenges in even identifying what the artist requires". The author emphasises that the requirements by the artist might change repeatedly until he/she is satisfied. Similarly, Marchese reports changes in requirements during the implementation of Trigger – "the system design was updated to reflect experiments with different types of sensors" (Marchese, 2006). Biswas and Singh (2006) share their experience from two installation projects, stating that often “the emergent system specifications cannot be defined in sufficiently tangible terms till the very end of the project”, especially because they might be very vague at the beginning. The reason for that might be due to the different working style of the artist – more exploratory rather than rationally planned and with explicit goals, as it generally is in business domain.

Experimentation as working style and the need of artistic control over a large set of easy-to-use functionalities are underlined as important requirements in the software development in interactive installation art (Edmonds, Turner and Candy, 2004; Biswas and Singh, 2006; Marchese, 2006).

As earlier stated, the installations that we examine are most often highly interactive. What differs from the common interaction in software systems is the final goal. In IS, the goal is to increase the effectiveness and the efficiency in the correct completion of a concrete task (or set of tasks). On the other hand, “humour and play are important aspects of the art” (Steinkamp, 2001). In addition, often ‘the system usage context is entirely absent or it is not well understood’ (Biswas and Singh, 2006). Hannington and Reed (2002) state that the difficulties in “capturing human activities in a manner that is sufficiently informal for non-programmers to understand, yet sufficiently precise for developers to use as a specification” are stronger in multimedia domain than in other domains. Interactive installations are often used by a large number and variety of spectators – adults and children, people with different education and knowledge, men and women of different cultures. Thus, “requirement elicitation should encompass sufficiently large variety of usage situations” (Biswas and Singh, 2006).

Several authors (Machin, 2002; Edmonds, Turner and Candy, 2004; Biswas and Singh, 2006) suggest that some of the difficulties in requirements elicitation might be due to practical communication problems (i.e. the two communities – artists and technologists – use their domain specific language, terms and concepts and might misunderstand each other or underestimate the importance of issues from the other domain). Biswas and Singh (2006) states that

> “technologists are forced to play a very limited role unless they had sensitised and educated themselves about non-technical design and contextual issues.”

The multidisciplinarity of this field requires openness from all participants towards the other domains.

It is important that both software developers and artists are aware of these properties of the requirements. Requirements might be difficult to capture, vague at the beginning and frequently changeable. Having this in mind will allow choosing the most appropriate software development methods, designing the most suitable architecture of the product, good risk assessment and proper planning of budget and schedule.

The literature review shows also that the software developers have to be an active side in the requirements definition when working with artists. In many cases, artists have clear ideas of what they want the final effect of the artwork on the audience to be. They might have also decided on what technology they want to explore. However, they might not be aware of the full potential of this technology and how it might influence on what the system will do. They expect suggestions and proposals from the technologists on what the technology allows. These ideas would not be directly applied, but would provoke/inspire the artist’s creativity and will be put together with his/her ideas and goals for the final artefact – the artwork. An artist might want the software components of the work to be part of the ideas and 'ethics' of the work and not only tools to reach a certain funcitonality.

5.2 Software architecture and design

The software architecture depends on the functionalities which should be provided by the system, on the quality attributes required and identified in advance, on the technology chosen, on designers' preferred styles, etc. Thus, the software architecture will most probably differ from one project to another.

For several of the interactive installations the software architecture is reported. Marchese (2006) describes a simple architecture with three components – microcontroller-sensor system, application software and an interface software between sensors and the application with "high level interrelationships among components without specifying the processing details". Boyd, Hushlak and Jacob (2004) report a pipeline architecture where "the output of a module can provide input to one or more other modules in a pipeline". Several standard software (i.e. previously available software not developed by the authors) were combined, including the software for simulating a swarm and a video interaction server widely used for surveillance tasks. The pipeline is made dynamically configurable through a graphical interface.

The software behind the interactive installation 'Swimming across the Pacific' (Fels et al., 2005) has also a modular architecture. The use of object-oriented software engineering methods is reported in Strömberg, Väätänen and Räty (2002). Machin (2002) describes an especially designed simulator and a specific language that allows the artist to easily experiment with the installation design and several supplementary tasks (e.g. calculation of the overall cost which depends on the changes of the materials used and their quantity). Similarly, Biswas and Singh (2006) have developed a mobile experience engine which helps in the simulation of the final artwork. Their system contains two parts – a visualiser that generates low-fidelity prototype and a code generator that generates high-fidelity prototype with optimised implementation for several interaction devices. The authors find that most suitable is to "wisely splitting the application architecture into two parts, one dealing with interactivity, the other tackling core functionality". Finally, Edmonds, Turner and Candy (2004) state that "In art and technology environments, we need environments for building environments".

The observation of the published work on interactive installation art shows that whenever possible the development teams tend to use software that is already available, and the software produced is often used (either completely or partially) in several consecutive artworks (see reuse in Section 3, point 6). This decreases the overall effort for implementation and the final price of the software. However, in most of the cases the standard components have to be integrated into the full system, custom parts have to be implemented and/or modified for the currently developed artwork.

Although artists often have much more profound technological knowledge than expected from clients in software engineering projects, they often would like to have the freedom of experimenting with all technological possibilities by themselves even in cases when their experience is not enough. Adding an extra layer between the artist and the underlying programming fosters the artwork creation without limiting the artist's creativity.

As mentioned before, experimentation and artistic control over technology area major challenges, reported by some of the authors (Machin, 2002; Edmonds, Turner and Candy, 2004; Biswas and Singh, 2006; Marchese, 2006). Constant verbal communication and negotiation is crucial and prototyping might be very helpful. The acquisition of domain knowledge and learning about the art domain by software engineers, however, might not resolve difficulties of vague and changing requirements in art projects, due to the experimenting nature of artists working styles. Artistic control might be increased by architectural approaches. Modular architectures allow easy changes of parts of the system and simplify the reuse of components when necessary. Investing more effort in creating easy-to-use graphical interfaces which allow rich experimentation with the full system might prove a winning strategy in this domain. Further research is needed to fully understand how to support such experimentation without limiting the creative process, but support it with proper software tools.

5.3 Evaluation, validation and testing

Software products are usually checked for their correctness during execution (i.e. testing), for satisfying the requirements specifications (i.e. validation) and on how well the end-product satisfies the user expectations (i.e. evaluation).

The testing might be done automatically by using other software that executes (pieces of) the system with various parameters and controls the correctness of the outputs. It might be also done manually by the developers and/or users, which is commonly done in small projects. For example, Marchese (2006) reports such manual approach for the integration testing of the interactive installation Trigger – "the developers systematically walked through the space triggering all video and sound sequences", thus simultaneously testing the hardware (e.g. sensors) and the software of the artwork. This, together with the validation was done on-site when the installation was mounted in the gallery several days before opening – "multiple walkthroughs of the installation by the artist before the opening constituted the final acceptance test of the system".

Strömberg, Väätänen and Räty (2002) report the use of heuristic expert evaluation for the technical aspects of the system. Multiple (iterative) evaluations were performed in different phases of the design and implementation with participants that were considered potential final users. The evaluation was done by observation, interviews and open-ended questionnaires and in earlier stages the feedback was used for design improvements. Fels et al. (2005) evaluated their artwork with the visitors of Siggraph 2004 exhibition, collecting opinions, positive and negative experience, comments and suggestions.

The evaluation against the final user utilisation might be essential for creating the user-system interaction as it is planned and expected by the artist. For example, Steinkamp (2001) reports that "children immediately understand that they are expected to play in the projection". On the other hand, adults were examining and analysing the system instead of actively interacting with it. This was not exactly the behaviour/effect desired by the artist, but it was noticed only when observing the audience during the exhibition. One might suggest that testing and evaluation of an art projects should include test audiences to be able to assess the quality of experience in the work as well.

While Hannington and Reed (2002) affirm that "most multimedia process models advocate use of strict evaluation and revision within the iterative cycles of development" this might not be possible in all cases due to budget or other limitations. Furthermore, clear distinction should be done between testing and evaluation, as "testing ensures correct technical operation of the system, but it does not ensure its appropriateness or its effectiveness in delivering the expectations." (Biswas and Singh, 2006)

Nevertheless, in some cases the testing of certain system parameters might be done in real-world situation – for example the robustness of the innovative face-detection software behind the '15 sec of fame' (Solina, 2004) was tested during the exhibition and based on participants evaluation the authors judged the algorithm as reliable and effective.

Summing up, the development team in interactive installation art project should consider evaluating the artwork and especially the interaction as early as possible with final users, as the effect might not be as expected by the artist. Different users should be considered – culture, gender, age, etc. Some of the quality attributes, such as reliability, robustness, etc. might be tested in real environment during exhibition.

5.4 Process models and project management

The software development model is a formalisation of the activities and the modes in which the software development is organised. Generally, this is part of the policy which the software development company has incorporated and is valid for all the projects within the company. Many interactive installations are developed during artists-in-residence programmes and the software development process might be influenced by the hosting institution practices.

Agile method of software development (i.e. adaptive software development) was chosen at the beginning of the project described in Marchese (2006) and was evaluated as a good choice by the software engineers. The developers predicted the possibility of vague requirements which would change frequently. The chosen method was suitable also because it dealt well with the strict schedulers of both artist and technologists and with budget limitations.

Biswas and Singh (2006) discuss several possible software development methods and their advantages and disadvantages for interactive art projects. For example, they state that "Creative artist's work processes do not necessarily follow 'analyse-model-design-build' trajectories like engineers (AN: see the waterfall model described in Section 3, point 4). They (artists) iteratively and intuitively generate creative ideas and evolve their design based on their perception and experience." More suitable from the traditional software engineering approaches is found to be the 'evolutionary prototyping' in which "artists will generate creative ideas, technologists will receive briefing from artists, build prototypes, elicit modifications/corrections and further requirements from the artists and this cycle will be repeated till the artist is progressively satisfied." However, this was far from the perfect model, as "system developed by evolutionary prototyping may suffer from lack of coherency in its architecture due to inadequate planning". The authors build upon this model and enhance it with low-fidelity and high-fidelity prototype generation. Biswas and Singh also suggest the need of further investigation of other methods, such as user assisted prototyping, participatory prototyping and prototyping combined with usability studies.

While in the previously discussed cases the development of the artwork was rather isolated from the final users, in other projects different approach has been chosen. Human-centered design has been used in Strömberg, Väätänen and Räty (2002). The authors report that the users were not only participating in the final evaluation, but also were "essential part of the design process from the early stages". This process is iterative and the necessary changes were made according the feedback from the users. The application functionalities were designed by the artists in the form of scenarios and storyboards that were used by the software designers for deriving an object-oriented architecture.

Table 3 Software tools

Software	*Description*	*C/OSS*[a]
3D Studio Max Strömberg (Väätänen and Räty, 2002)	3D modelling and animation package www.autodesk.com/3dsmax/	C
Breve swarm simulation (Boyd, Hushlak and Jacob, 2004)	A package for building 3D simulations of multiagent systems and artificial life www.spiderland.org/breve/	OSS
GigaStudio 160 (Strömberg, Väätänen and Räty, 2002)	Software for music and sound effects www.tascamgiga.com/	C
Macromedia Director (Sardon, 2006)	Multimedia authoring tool www.adobe.com/products/director/	C
Macromedia Flash (Strömberg, Väätänen and Räty, 2002; Marchese, 2006)	Professional software for creating rich, interactive content for digital, web and mobile platforms www.macromedia.com/software/flash/	C
MAX/MSP (Bestor, 1996; Edmonds, Turner and Candy, 2004; Marchese, 2006; Satomi and Sommerer, 2007)	A graphical environment for music, audio and multimedia www.cycling74.com/	C
Maya (Steinkamp, 2001)	3D animation package http://www.autodesk.com/maya	C
Mobile Bristol toolkit (Biswas and Singh, 2006)	A tool to create and share mobile, location–based media www.mobilebristol.com/	OSS
Mobile experience engine (Biswas and Singh, 2006)	A software development platform for creating advanced context-aware applications and media-rich experiences for mobile devices www.open-mee.org/	OSS
OpenGL (Fels et al., 2005)	Industry standard and API for high performance graphics www.opengl.org/	OSS
Particle dynamics (Steinkamp, 2001)	A software tool set for simulating natural phenomena.	
Pfinder (Sommerer and Mignonneau, 1999)	Advanced camera/gesture tracking software developer MIT, vismod.media.mit.edu/vismod/demos/pfinder/	
Sculpture simulator (Machin, 2002)	A sculpture simulator with its own programming language	
SoftVNS video toolkit (Edmonds, Turner and Candy, 2004)	A real time video processing and tracking software for MAX/MSP homepage.mac.com/davidrokeby/softVNS.html	OSS

[a] The commercial products are marked with C. The free or open source products are marked OSS. For the rest this information is not available.

Considering the whole project management, several issues should be mentioned. In interactive installation art, often the creation of the artwork is sponsored by a public entity which does not influence on the artist's creativity. However, the budgets are tight and often relatively small. In many cases, the work is created during artists-in-residence programmes and it is possible that some members of the team have additional work duties. Commonly, there is an opening day for the artwork, thus the final deadline for the full system might not be flexible. Apart from budgeting and scheduling probably most important issue in the project management is the risk assessment. According to

Marchese (2006), “any component of the system or member of the project could pose a risk”. Artists often explore and incorporate in their interactive art installations one or more new/emergent technologies (e.g. sensors, location awareness, mobile and wireless, etc.). Together with mostly limited budget, this leads to the high probability that the technologists will not be well acquainted with the necessary skills and need time to learn. The newest technology might also be problematic in terms of instability, lack of documentation and support materials and communities.

5.5 Development environments and tools

In Table 3, we provide a list of the software tools mentioned in the reviewed articles, together with additional information and links.

The development tools used by software developers are not discusses in the reviewed articles. Our guess is that standard CASE tools were utilised (the ones which the technologists are most used to) with no particular advantages or disadvantages for the software development in interactive installation art. Although artists sometimes prefer “access to deeper levels of the computer’s programming system” (Edmonds, Turner and Candy, 2004) the tools that are suitable for software engineers does not seem to be proper for them. MAX/MSP is distinguished as a very suitable tool for artists in installation art (mainly for the musical side, but not limited (Bestor, 1996; Edmonds, Turner and Candy, 2004; Marchese, 2006; Satomi and Sommerer, 2007). Interestingly, they use tools like Macromedia Flash as CASE tools – for implementing their programmes, as reported in Marchese (2006), but also for supportive tools, such as for creating the storyboard in Strömberg, Väätänen and Räty (2002). In several cases, e.g. (Machin, 2002; Edmonds, Turner and Candy, 2004; Biswas and Singh, 2006), technologists provide additional software layer for the artists – tools that will give them the freedom to experiment without limiting their creativity. Such tools are found to be well accepted and positively evaluated by artists. Some suggestions and guidelines about the future of the creativity support tools might be found in Shneiderman (2007).

6 Discussions

In the previous sections, we have synthesised the basic software engineering concepts, we have shown how interactive installations might be mapped to software engineering products and we have summarised practitioners’ experiences reported in the literature.

Many of the problems, difficulties and issues we observe in installation art projects are similar to the ones commonly met in contemporary software development in other domains. However, we can identify several characteristics which are peculiar for artistic projects.

Experimentation as a working style of artists. This is a major challenge and further research is needed to fully understand how to support such experimentation without limiting artists’ creativity. As mentioned, constant verbal communication and negotiation is crucial and prototyping might be very helpful. The acquisition of domain knowledge and learning about the art domain by software engineers might not resolve difficulties of vague and changing requirements in art projects, due to the experimenting nature of artists working styles. On the other hand, we observe that artists are interested in gaining software engineering knowledge, which, in our opinion, is not typical for clients in other domains. The implications of this particularity might be in different directions. For example, software engineers might need to redefine some of the traditional software engineering concepts and methods, e.g. requirements elicitation. The need of tools that assist, enable and empower creativity is noticeable. Research on the possibilities to increase the 'artistic control' over technology is urging.

Entertainment or reflection as a final goal. One of the main differences we observe in art-related projects is the application aim/goal. While in other domains like business, commonly the goal is to increase task efficiency, in art the goal is to communicate in an aesthetic way and to raise awareness in the audience about contemporary phenomenon and its implications for the future of the society. This characteristic leads often to the production of non-critical applications. Due to this, multidisciplinary projects might be very good for educational purposes – courses/projects involving software engineering students and artists/art students cooperating with a common final outcome are reported to be very well accepted (Jaccheri and Sindre, 2007). Meanwhile, unexpected (even incorrect) functionalities of the developed programmes might be considered by the artists as non-negative, interesting and inspiring effects. An implication to this particularity of installation art is that specific attention should be paid and further discussion might be needed on how to evaluate the final products (i.e. the artwork as a whole and the software in particular) and what are the success measurement in this context.

Ideological influence on the choice of technology. Artists' preference for utilisation of (emerging) technologies such as mobile devices, surveillance cameras, sensors, etc. is often due to ideological motivation – they are driven by their desire to provoke reflection in the audience. The technological choice influences on the use of software technology as well. This differs from other domains, where the choice of software technology is based on the workflow processes that the new software has to support and the needs to integrate new systems with previous available systems. In addition, we observe ideological choice even of the software technology: for example, artists use open source software for ideological reasons. What typically happens, according to our personal observations, is that when they try the product they find the user interface unfamiliar/difficult and some features they are used to and they need are missing. Thus, they switch back to proprietary applications, such as Final Cut Pro/Acid/Photoshop. Open source software is not only a free of charge basis for software projects, but also provides a community support and further development which artists might explore. Further investigation on these questions is needed. A challenge for software engineers (SE) is to keep the artists' interest in participating in open software development, as their artistic and innovative input of ideas for new functionalities and solutions to problems might prove to be very valuable.

Artists' *tendency to utilise a wide variety of mostly emerging technologies.* In interactive installation art projects, we often observe the use of software to support large variety of functionalities (e.g. control of sensors, interaction, content generation, web presentation, etc.). The software is often developed through a single project by a small team with generally tight budget and schedules. This implies the need of cautious choice of software development methodology and project management, including careful risk assessment. Furthermore, the reuse of software components is a common target-domain specific issue in software engineering, which might be worth exploring in art.

Throughout our literature study, we found out that important software engineering issues, like maintenance and open source software, are not mentioned in any of the reviewed articles. We ask if it depends on the fact that they are not important in interactive installation art or if there is some other reason why they are omitted. Our experience implies that they should be important. Artists often continue work on their interactive installations; they evolve and are exhibited in consecutive occasions. In some cases, the installation might be accessible to the public for a long period (for example one of our case studies – the interactive installation Flyndre in Inderoy, Norway, which will be exhibited for 10 years. See www.flyndresang.no/). Due to the long exhibition period, it is important that the software supporting different functionalities is maintained properly. Then, why is software maintenance not discussed in the literature? Our assumption is that this might be related to limited budgets of projects in interactive art installations. As we mention in Section 3, in other domains up to two-third of the software cost might fall on software maintenance.

We believe that further studies are needed to better understand artists' perspective on technical and non-technical issues in interactive installation projects. This article provides rather biased one-side view of interactive installation art. We have described the software engineering concepts and issues (in Section 3) in the language and with the terms utilised in software engineering. The literature review (Section 5) covers mainly scientific articles found as part of a systematic literature review. However, the systematic literature review showed to be an inadequate method for understanding artists' perspective. Using case studies, on-side observations and interviews with participants in relevant projects are expected to be much better research methods in this context. We have initiated research in this direction, however it is out of the scope of the current article.

7 Summary and conclusions

Interactive installation art is an active field with many artworks created and exhibited, both in Norway and worldwide. Frequently, interactive installations are heavily dependent on software and often software engineers are part of the team in projects for creating interactive installation. During our involvement in several such projects, we have asked the question: how much does the artist need to know about software engineering in order to interact with software engineers and programmers?

In this article, we outline the key software engineering concepts in relation to the development of interactive installation art. This will give an easy start for artists and will facilitate the creation of common language and understanding within the team. We also summarise the software engineering issues and solutions reported in published articles describing interactive installations. This collection of experiences from interactive installation art projects might guide the software engineers and programmers in their future practices and direct their attention to difficult issues in this specific domain. We also provide a list of utilised tools.

It is important that both software developers and artists are aware that in interactive installation art requirements are difficult to capture, vague at the beginning and frequently changeable. Software engineers have to be an active side in the requirements definition. The choice of appropriate software development method and the design of suitable software architecture might capture this specific of the domain and might help for the proper planning of the budget and schedules. Careful risk assessment should be performed to guarantee a successful ending of the project.

Interaction is a key issue in interactive installation art. Sometimes, the environmental parameters trigger the interactivity, sometimes only the audience presence is enough to cause changes in the artwork, but in most of the cases the interaction is directly linked to the audience actions. But how to understand the audience and how to predict their behaviour to the extent to reach the artist's desired effect? In Section 5, we have shown that different process models might be used to face this issue like human-centered design. Prototyping is found suitable in order to ensure correct understanding between artist and software developers (i.e. correctness in requirements specification). Prototypes might also foster the evolvement of the artist's creative ideas. In addition, the evaluation of the artwork interactivity might be done in different stages of the project by the expected audience. However, a large variety of spectators (i.e. gender, age, nationality, education, physical capabilities, etc.) might be anticipated.

It should be kept in mind that different stakeholders in interactive installation projects might need and prefer to use different set of tools for their tasks. Software developers are acquainted with CASE tools appropriate for the software design and implementation. These tools, however, might not be appropriate for artists. Artists often use tools such as Flash or Max/MSP as CASE tools. They might prefer to create storyboards instead of UML diagrams. In their work, artists often adopt a more experimental style and might require an additional software layer (e.g. a tool) that will allow them to experiment with the chosen technology without limiting their creativity.

Interdisciplinary research should provide benefits to all disciplines involved. In our case, the primary goal is that of providing guidelines for artists that need software engineering knowledge and a roadmap for software engineers to the existing (i.e. reported in the literature) relevant work. On the other direction, there are at least the following beneficial issues for software engineering:

1 *Education and recruitment of students*. Students are motivated when working on their art-related projects (Jaccheri and Sindre, 2007). They can touch with their hands their work and show it to family and friends.

2 *Cultural and social aspects*. Computer scientists and software engineers need to reflect about the nature of our discipline and on our profession The dialogue with artists and the concrete projects and problems are a stimulus to such reflection.

3 *Technological innovation*. Artists work in a different way and generate requirements that may result in innovative products and methods, e.g. Harris (1999).

From the artists' point of view, an awareness of the possibilities of applying different and more flexible software engineering methods, could give better artistic results and artworks where the software is conceptually better integrated in the work as a whole.

References

Bass, L., Clements, P. and Kazman, R. (2003) *Software Architecture in Practice* (2nd ed.). Boston, MA: Addison-Wesley Professional, ISBN: 978-0321154958.

Bestor, C. (1996) 'Max as an overall control mechanism for multidiscipline installation art', *Computers and Mathematics with Applications*, Vol. 32, pp.11–16.

Biswas, A. and Singh, J. (2006) ‘Software engineering challenges in new media applications’, in A.M.K. Cheng (Ed.), *Software Engineering Applications (~SEA 2006~)*. Dallas, TX: ACTA Press. 11/13/2006–11/15/2006.

Bourque, P., Dupuis, R., Abran, A. and Moore, J.W. (Eds) (2004) *Guide to the Software Engineering Body of Knowledge*. Piscataway, NJ: IEEE Press.

Boyd, J.E., Hushlak, G. and Jacob, C.J. (2004) ‘Swarmart: interactive art from swarm intelligence’, Paper presented in the Proceedings of the *12th Annual ACM International Conference on Multimedia*. New York, NY: ACM Press.

Cockburn, A. (2006) *Agile Software Development: The Cooperative Game* (2nd ed.). Boston, MA: Addison-Wesley Professional, ISBN: 978-0321482754.

Conradi, R. (2007) *Software Engineering Mini Glossary*, Available at: http://www.idi.ntnu.no/grupper/su/publ/ese/se-defs.html, last visited 19/06/07.

Digital Art Museum (2007) *History*, Available at: http://www.dam.org/history/index.htm, last visited 26/06/07.

Edmonds, E., Turner, G. and Candy, L. (2004) ‘Approaches to interactive art systems’, Paper presented in the Proceedings of the *2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*. San Francisco, CA: ACM Press.

Fels, S., Kinoshita, Y., Tzu-Pei Grace, C., Takama, Y., Yohanan, S., Gadd, A., Takahashi, S. and Funahashi, K. (2005) ‘Swimming across the pacific: a vr swimming interface’, *Computer Graphics and Applications, IEEE*, Vol. 25, pp.24–31.

Fujimura, N. (2004) *Remote Furniture: Interactive Art Installation for Public Space. ACM SIGGRAPH 2004 Emerging Technologies*. Los Angeles, CA: ACM Press.

Hannington, A. and Reed, K. (2002) ‘Towards a taxonomy for guiding multimedia application development’, *Ninth Asia-Pacific Software Engineering Conference (APSEC’02)*, pp.97–106, Gold Coast, Queensland, Australia, December 4–6.

Harris, C. (Ed.) (1999) *Art and Innovation: The Xerox Parc Artist-in-Residence Program*. Cambridge, MA: MIT Press.

Jaccheri, M.L. and Sindre, G. (2007) ‘Software engineering students meet interdisciplinary project work and art’, *11th International Conference on Information Visualisation (IV)*, Zurich, Switzerland, July 2–6.

Lysaa, P.A., Sandboe, Y. and Kvinnsland, B. (2006) ‘Intravision art presentation’, *Intravision Presentation, March, 2006*, Available at: www.intravision.no/downloads/iv_art_2006.pdf (13/06/07).

Machin, C.H.C. (2002) ‘Digital artworks: bridging the technology gap’, Paper presented in the Proceedings of the *20th Eurographics UK Conference*.

Manovich, L. (2002a) *The Language of New Media (Leonardo Books)*. Cambridge, MA: MIT Press, ISBN: 978-0262632553.

Manovich, L. (2002b) ‘New media from borges to HTML’, in N. Wardrip-Fruin and N. Montfort (Eds), *The New Media Reader*. Cambridge, MA: MIT Press.

Marchese, F.T. (2006) ‘The making of trigger and the agile engineering of artist-scientist collaboration’, Paper presented in the Proceedings of the *Conference on Information Visualization (IV)*. IEEE Computer Society.

Naur, P. and Randell, B. (1968) ‘Software engineering: report of a conference sponsored by the nato science committee’, in P. Naur and B. Randell (Eds), *Software Engineering*. Garmisch, Germany, 7–11 October, Brussels, Scientific Affairs Division, NATO (1969), 231.

Oates, B.J. (2006) ‘New frontiers for information systems research: computer art as an information system’, *European Journal of Information Systems*, Vol. 15, pp.617–626.

Østerlie, T. and Jaccheri, L. (2007) ‘A critical review of software engineering research on open source software development’, Paper presented in the Proceedings of the *2nd AIS SIGSAND European Symposium on Systems Analysis and Design*, Gdansk, Poland, June 5.

Østerlie, T. and Wang, A.I. (2006) 'Establishing maintainability in systems integration: ambiguity, negotiations, and infrastructures', in R.K.A.D. Binkley (Ed.), Paper presented in the Proceedings of the *22nd IEEE International Conference on Software Maintenance (ICSM'06)*, pp.186–196, Philadelphia, Pennsylvania, USA, IEEE CS Press, September 25–27.

Østerlie, T. and Wang, A.I. (2007) 'Debugging integrated systems: an ethnography of debugging practice', in L. Tahvildari and G. Canfora (Eds), Paper presented in the Proceedings of the *23rd IEEE International Conference on Software Maintenance (ICSM 2007)*, pp.305–314, Paris, France: IEEE CS Press, October 2–5.

Pressman, R.S. (2004) *Software Engineering: A Practitioner's Approach*, (6th ed.). McGraw-Hill Science/Engineering/Math, ISBN: 978-0073019338.

Project Management Institute (2004) *A Guide to the Project Management Body of Knowledge*' (3rd ed.), Project Management Institute. ISBN: 978-1930699458.

Sardon, M. (2006) 'Books of sand', Paper presented in the Proceedings of the *14th Annual ACM International Conference on Multimedia*, Santa Barbara, CA: ACM Press.

Satomi, M. and Sommerer, C. (2007) ''Game_of_life': interactive art installation using eye-tracking interface', Paper presented in the Proceedings of the *International Conference on Advances in Computer Entertainment Technology*, Salzburg, Austria: ACM Press.

Shneiderman, B. (2007) 'Creativity support tools: accelerating discovery and innovation', *Commun. ACM*, Vol. 50, pp.20–32.

Solina, F. (2004) '15 Seconds of fame', *Leonardo*, Vol. 37, pp.105–110.

Sommerer, C. and Mignonneau, L. (1999) 'Art as a living system: interactive computer artworks', *Leonardo*, Vol. 32, pp.165–173.

Steinkamp, J. (2001) 'My only sunshine: installation art experiments with light, space, sound and motion', *Leonardo*, Vol. 34, pp.109–112.

Strömberg, H., Väätänen, A. and Räty, V-P. (2002) 'A group game played in interactive virtual space: design and evaluation', Paper presented in the Proceedings of the *Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, London, UK: ACM Press, June 25–28.

Tribe, M., Jana, R. and Grosenick, U. (2006) *New Media Art (Taschen Basic Art)*. Taschen America, LLC: ISBN: 3822830410.

Trifonova, A., Ahmed, S.U. and Jaccheri, L. (2007) 'Sart: towards innovation at the intersection of software engineering and art', Paper presented in the Proceedings of the *16th International Conference on Information Systems Development*, Galway, Ireland, Springer, August 29–31.

Wiegers, K.E. (2003) *Software Requirements* (2nd ed.). Redmond, WA: Microsoft Press, ISBN: 978-0735618794.

Notes

[1]ARS Electronica – festival for art, technology and society – is one of the most popular annual events on art and technology. As part of it, since 1987 prices are given to the best artworks in different categories (Prix Ars). The evolvement of these categories shows also the changes in the trends in the domain (see www.aec.at/en/prix/). Interactive art, including installations, is one of the categories. It first appeared in 1990.

[2]We often talk about the artist and/or developer (in singular) for simplicity, although there might be cases where a group of artists/developers (two or more) are working together on the same artwork, either simultaneously on the whole work or on different parts of it (e.g. one artist on the music components, another on visualisation; one software engineer on the software architecture and another on implementation, etc.).

[3]The term 'evolutionary' is used by the authors of the cited article as reference to evolutionary image processes, as their works are bio-inspired.