Software engineering for and with artists: a case study

Anna Trifonova
Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
7491 Trondheim, Norway
trifonova@idi.ntnu.no

Øyvind Brandtsegg
Department for Art and Media Studies
Norwegian University of Science and Technology (NTNU)
7491 Trondheim, Norway
obrandts@gmail.com

Letizia Jaccheri
Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
7491 Trondheim, Norway
letizia@idi.ntnu.no

ABSTRACT
The article presents ImproSculpt – a live performance instrument for algorithmic composition and improvised audio manipulation. A custom version of the software has been used within several interactive art installations, one of which (i.e. Flyndre) will be described. Besides the software characteristics and usage modes we present and discuss the software engineering challenges, problems and lessons learned during the development of ImproSculpt. Several methodologies were used in the case study: observing the project through its life cycle; analyzing the software and its documentation; questionnaire with the artist who is also the main developer; two software engineering interventions. The support of high performance, easy modifiability and availability were found to be particularly important. The development of a modular architecture has been identified as a way to satisfy some of the non-functional quality attributes of the software that appeared with the growth of the project. Furthermore, ImproSculpt has been published as open source software in order to increase access to wider public and stimulate input from interested community - software developers and artists.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]: General; J.5. [**Arts and Humanities**]: Performing Arts.

General Terms
Design, Documentation, Human Factors, Management.

Keywords
Software Engineering; New Media Art; Algorithmic Composition; Improvised Audio Manipulation.

1. INTRODUCTION AND BACKGROUND
Since the birth of digital technology it has been utilized within the production of art. The personal computer, continuous decrease of prices and increase of computational power had led to widening of the application domains and the number of uses.

Software engineering studies the design, the development and the maintenance of software. The goal is to understand the current practices and needs in different domains, analyze them and propose improved processes and/or tools. At the Norwegian University of Science and Technology (NTNU) a group is dedicated on studying Software Engineering issues in art domain - SArt project, http://prosjekt.idi.ntnu.no/sart/. Members of SArt participate and observe several artistic projects in which software is developed, including ImproSculpt by Øyvind Brandtsegg.

Originally from a jazz improvisation background, Brandtsegg has always had a strong desire to experiment and discover new ways of making music. He is a musician and composer who takes a special interest in electronics and computers related to the making of sound. Since 2000, he started the development of ImproSculpt - a tool for algorithmic composition and improvisation, which we describe in this article. The project continues to develop up to date and the software has been used in many artworks – performances, CDs, installations. As the software grew and its functionalities increased it has attracted many artists.

With the increasing of the size and the complexity of ImproSculpt, the need of better software engineering became obvious. Several interventions were performed by SArt members. Software engineering students from NTNU (Thor Arne Gald Semb, Audun Småge, Thibault Collet, Maximo Ramirez, Adrien Garioud, Christophe Lebegue, Godfrey Mayende, Henrik Mogens Gundersen, Mari Lie Hæreid and Trond Engu) have developed or improve aspects of ImproSculpt from software engineering point of view. The software architecture has been re-designed to make it modular, easy to extend and modify [1]. In addition, ImproSculpt has been published as open source [2]. Furthermore, members of SArt have observed the development of ImproSculpt through its lifecycle and used a questionnaire for understanding the main challenges. Further in the paper we discuss the lessons learned from this project which will be useful to artists and software developers that develop software for art purposes.

In the following sections we describe the overall goal, functionalities and methods used in ImproSculpt (section 2). We also give some details on the design and development choices, the tools and languages utilized (section 3). Finally, we make a reflection and share the lessons learned from this software development project (section 4). A section dedicated to the related work (5) is followed by a summary (6) and references (7).

2. SOFTWARE CHARACTERISTICS
Here we will present the software and its functionalities. ImproSculpt Classic (2.1) is the first version developed by the

artist Øyvind Brandtsegg in Csound. It has earned to its developer much respect, especially in the Csound community. It has been later re-designed and re-developed using Python and Csound, where Python acts as a host to Csound. ImproSculpt4 (2.2) is the current major version of the software with added functionalities. A custom version of ImproSculpt is utilized for the interactive sound of Flyndre (2.3) - an art installation in Inderøy, Norway.

ImproSculpt is software for live sampling and manipulation, algorithmic composition and improvised audio manipulation in real time. It might be considered also a live performance instrument. Brandtsegg has used it in a variety of performances and studio settings, collaborating with different musicians.

"Most of the time, the instrument might be able to bring you surprises... bringing in a new musical element. You are never 100% in control, it is more like you push things in a general direction, let it evolve, and then adjust the bits you do not like and refine the bits that already sound good." (Øyvind Brandtsegg)

At the time of writing, ImproSculpt has been used across the globe as both a compositional tool and a live performance instrument, e.g. [3] and [4]. Several recordings can be found the composer's web site [5].

The individual algorithms used in ImproSculpt are not very complicated, but the system gains complexity through combining all of these in various ways. Typically, the most interesting results are achieved by combining several modules and effect filters.

The ideas behind ImproSculpt, the main implementation of code and the ongoing development of the system are mainly done by Øyvind Brandtsegg. In addition, several software engineering students from NTNU have participated and helped throughout the project as part of their study and theses [1, 2, 6].

2.1 ImproSculpt Classic

ImproSculpt Classic was the first version of the software, which has been developed entirely by the artist.

In 2000, Brandtsegg started working on the original ImproSculpt. It was developed in part as a compositional tool for a commissioned work from by Midt-Norsk Solistensemble (a vocal choir). The commission was not specifically for the software, but for a musical piece (in 2000). The artist wrote the software to enable the performance of the piece, which allowed him to realize some of his prior ideas. Computers were becoming powerful enough to handle a lot of the tasks Brandtsegg had visualized in software, and a year later, the choir work was performed with the first version of ImproSculpt.

ImproSculpt consists of the following functionalities:

- *Sampling:* recording of new sound material into ImproSculpt, usually through a microphone, but any sound source can be used, even a feedback loop.
- *Rhythm control:* a process of triggering events in the program, based on the tempo and time signature dictated by the user. Specific triggers can also be programmed.
- *Compositional logic:* samples, or in some cases other sound material, are used to produce sounds. Several kinds of compositional techniques are implemented.
- *Effect processing:* The application of filters to the sounds from the compositional modules, such as distortion, reverb and equalizer envelopes.
- *Sound rendering:* The audio from the main ImproSculpt mixer is sent to the platform running Csound, which typically routes it to the soundcard and subsequently loudspeakers.

The graphical user interface (GUI) of ImproSculpt lets the user interact with all parts of the process. The code structure of the Classic version is flat, consisting of a single large Csound script file with nearly 5500 lines of code. There are no distinct classes or objects, other than those inherent in Csound itself.

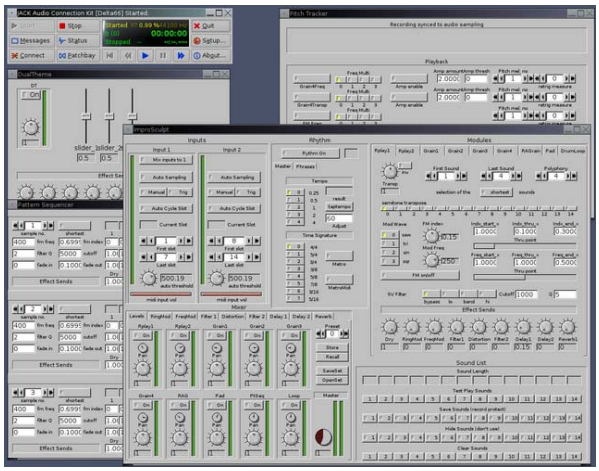

Figure 1: ImproSculpt (Classic) interface
(source www.linuxjournal.com/node/1000181)

The classic version has also been used for several art installations, for example TONO's "lydløype" 2003, "Hermetrollet" at Ringve Museum (Trondheim), "Irrganger" (Labyrinths) at Steinkjer videregående skole (Steinkjer College). For each work a custom version of the software was developed and the new functionalities were incorporated into the current development version.

2.2 ImproSculpt 4

As the original ImproSculpt project matured, more features were continuously added, and the source code eventually grew large and unruly. Brandtsegg discovered that the task of adding new features became increasingly strenuous, as virtually the entire system was affected by even minor adjustments to the source code. Eventually frustrated with the amounts of "collateral" work associated with further development of the system, Brandtsegg decided to give up developing the old source code any further, and start from scratch with a new and more structured approach.

Brandtsegg states that he "never planned for the project to become so big", which is why there was no long-term planning of architecture or program structure (i.e. it just evolved from several smaller sound modules). It has become nigh-on impossible to add new modules and composition techniques to the program, since any small change depends on changing a lot of different code.

Brandtsegg started the work on developing a new version of ImproSculpt in 2004, with more modular structure, and plans for increased modifiability. The aim was to separate parts for sound synthesis, mapping, compositional algorithms and user interface, in such a way that a change in one place does not necessitate changes elsewhere. Øyvind Brandtsegg rewrote the ImproSculpt software in Python programming language with an interface to Csound, instead of only utilizing Csound's native opcodes.

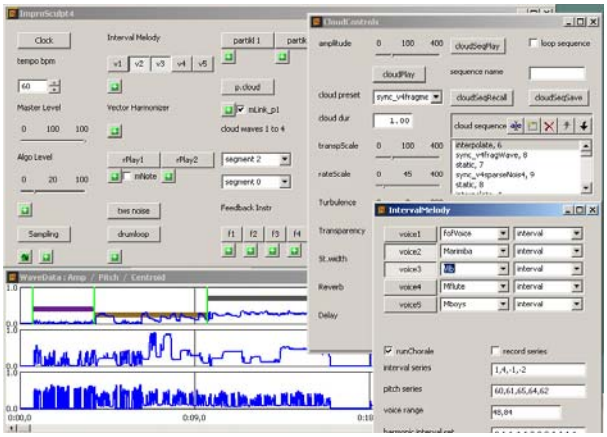

Figure 2: ImproSculpt 4 interface

At an early stage of the development of the new ImproSculpt, two NTNU master students intervened by analyzing the functionalities and the system's implementation and proposing improvements. ImproSculpt architecture has been re-designed according to predefined quality attributes (see sections 3.1 and 3.2)

The architecture has been further developed by Brandtsegg since the end of the students' project. New composition modules have been added, and the modularisation have been further extended. The gui and the audio processing code have been split into separate applications, communication via network interface, and these two parts of the system can now be run on separate computers if needed. The code for the GUI has also been modularized and generalized to enable more reuse of code. New audio processing methods have been implemented, and an extensive midi control mapping included.

Currently, ImproSculpt 4 contains about 26 000 lines of code (both Python and Csound) from which about 95% are new code and 5% are reused old code (with maybe some refurbishment).

2.3 Flyndre

Flyndre (www.flyndresang.no) is an interactive art installation. It is a sculpture located in Inderøy, Norway. On Figure 3 one can see the physical construction of the artwork which is placed in the natural site. The physical sculpture was made by Norwegian artist Nils Aas in 1999, and Aas did respond positively to Brandtsegg's request (in 2003) for adding a sound installation to the existing sculpture.

People can walk around the sculpture or sit nearby to watch it and listen to its music. It has an interactive sound system that has the goal to reflect the nature around the sculpture. To implement this goal the sound changes depending on parameters like the local time, light and water level, temperature, etc. The parameters from the environment are captured by sensors and influence the generation of the music. These parameters are integrated by interaction rules into the music which is played by the sculpture continuously. Some of the rules are static (i.e. fixed results are produces by certain parameters), while other rules are dynamic (e.g. algorithmic composition). The 'content' in this interactive installation is the music. It includes samples pre-defined by the composer, which are combined with and overlaid by the dynamically generated sounds. The composition techniques involved was designed to allow the software to take musical choices on behalf of the composer, taking into account the input parameter values at the time of decision. Such musical decisions are run on a per minute basis in the software.

Figure 3: Flyndre installation, Inderøy, Norway

Flyndre relies on software in several of its main parts – 1) for controlling the numerous sensors that are capturing environmental parameters, 2) for incorporating the captured parameters into music that is played by the sculpture (ImproSculpt), 3) for maintaining an archive of the music generated in the past, 4) for online presentation of the artwork, the artist, the music archive and the software. The web site (part of it in HTML and part in Flash) is an important part of the whole project and includes an on-the-fly animated Flash application that displays the current parameters of the environment and plays a live audio stream from the sculpture. The archive of music previously played by the sculpture is also accessible through the web site in MP3 format.

At the controlling core of the sound installation there is a custom version of the software ImproSculpt. The artist has started the software project that implements the functionalities of Flyndre. He has been the sole responsible for development of the modules which implement the composition logic (e.g. the sampling algorithms or the integration of particular sounds which are generated based on the environmental parameters). The generated music reflects the composer's aesthetical values and/or the desired [by the artist] effect/impression on the audience.

Students from NTNU, as part of Experts in Team course [6] in collaboration with Brandtsegg and Soundscape Studios[1] (Trondheim), have developed the technical framework for the networking and the sensors system - for capturing sensors data and transferring it via the Internet to the sound processing station.

3. SOFTWARE ENGINEERING ISSUES

Here we will give an overview of the software engineering issues and approaches in ImproSculpt development. Two major interventions had been introduced: re-design of the software architecture and publishing the software as open source. Why such interventions have been necessary and how they had influenced on the software development will be discussed.

[1] www.soundscape-studios.no

3.1 Requirements

Earlier we have mentioned that the first ImproSculpt version was developed by the artist Øyvind Brandtsegg for the creation of a commissioned musical piece. The functional requirements have never been formally defined. The artist had implemented his artistic ideas in a program without following software engineering principles, as it was not expected to grow into such a complex system. The informal definition of the functional requirements is a continuous process, as it follows the artists' constantly evolving ideas and experiments with artistic expression. Currently, the functional requirements are partially defined and written in a 'to do' list. Many of them, however, are in the artist's head. These are artistic ideas which are difficult, if not impossible, to elicit into detailed formal requirement which might be implemented by external developers (i.e. other than the artist himself).

As the system became more complex it became obvious that a long-term planning is required. Through a series of discussions, informal meetings and more formal interviews with the artist two NTNU students analyzed the non-functional requirements. They have used a "miniature version of Architectural Trade-off Analysis Method process[2]" [1]. In order to modify the architecture they have prioritized the quality attributes according to their importance for ImproSculpt (performance, modifiability, availability, usability, testability, security, safety). The prioritized list was agreed with the artist.

The requirements are taken directly from Brandtsegg's expressed wishes and needs for ImproSculpt.

R1 The system must be able to sample, process and play back sounds in real-time;
R2 Sounds must be able to be processed by any or all of the modules at any time;
R3 The system must be easily expandable with new sound processing modules;
R4 The system must be easily maintainable;
R5 The system user interface must be efficient.

The quality requirements are based on the above listed requirements, but through interpretation and reasoning, they have been concretized and divided into more specific requirements primarily belonging to one of the several quality attributes. The quality attributes were prioritized and agreed with Brandtsegg. Further we list them from most important to least important.

The *performance* attribute deals with how long it takes for the system to respond when an event occurs (latency). It is also concerned with how much resources, such as processing power, memory and storage space the system consumes during its execution. Since ImproSculpt is a real-time processing system, this quality attribute has a very high priority.

QR1 The system must handle audio input, processing and output with very low latency;
QR2 The system must run well on today's market mid-range computers.

Modifiability concerns changes in the system. Changes are often divided into three subcategories: local, non-local and architectural. The local change is simply changing a single element of code, without having to adjust anything else. A non-local change has ramifications other places in the system without fundamentally changing the architecture, while architectural changes bring about major changes in the way the system works. A good modifiable system structure allows most changes to be local, as they are by far the least expensive.

QR3 The system must have a modular object-oriented design that is easy to understand;
QR4 Changes made to the system should be as local as possible (little to no ramifications for other parts of the system);
QR5 The system must be easily expandable with new processing modules;
QR6 Csound changes should not affect ImproSculpt drastically, or vice versa.

Availability is concerned with component errors, their handling, and whether they lead to system failures or not. A failure has occurred when the user does not receive results consistent with the specifications of the system, while errors/faults should largely be invisible to the user and handled internally. Faults often lead to failures, if they are not corrected and/or masked.

QR7 As the system will be used in live performances, it is of great importance that the system produces no fatal errors of such nature that the program halts or crashes (i.e. failures);
QR8 The system must be able to handle "wrong" or illogical usage. That is, it must be able to handle unexpected input from the user without crashing;
QR9 The system must be able to catch internal errors in the software in such a manner that the system never terminates because of these;
QR10 The system must have an error rate of under 1 per 100 time units, where one time unit corresponds to 10 seconds interaction with the system. An error is an action from the system that differs from the expected behaviour. The errors mentioned here are non-critical, in that they do not cause the system to halt, but may produce wrong or unexpected output.

Many of the above mentioned quality attributes (e.g. performance, availability) were already at a satisfactory level. In this context the students concentrated their efforts on improving modifiability. The tactics they used is of polarizing responsibilities into separate packages in the system, thus improving the object orientation and maintaining semantic coherence [1].

3.2 Software Architecture and Design

Currently, ImproSculpt has a modular architecture, which facilitates easy modification, code reuse and continuous extension of the system (see Figure 4).

The core application consists of a central event caller, communicating with the audio processing module (csound), the composition modules, and the modules handling timed automation. The gui application handles the graphical interface. The two applications communicate via Pyro (Python Remote Objects) network proxies, and can be run on separate computers.

The NTNU students' intervention consisted in re-designing the former software to follow the Model-View-Controller pattern. Although the architecture has evolved after this intervention into the current form shown on Figure 4, the artist has continued to follow the Model-View-Controller pattern. The View (i.e. the gui

[2] The Architecture Tradeoff Analysis Method (ATAM) http://www.sei.cmu.edu/architecture/ata_method.html

application) communicates with the Controller, represented on the figure by the eventCaller package. The Controller works as the central hub of the program and handles communication between the other packages. The composition modules represent the Model. When the Controller has received the data needed from the Model, the data can be sent back to the View. The View gives feedback to the user.

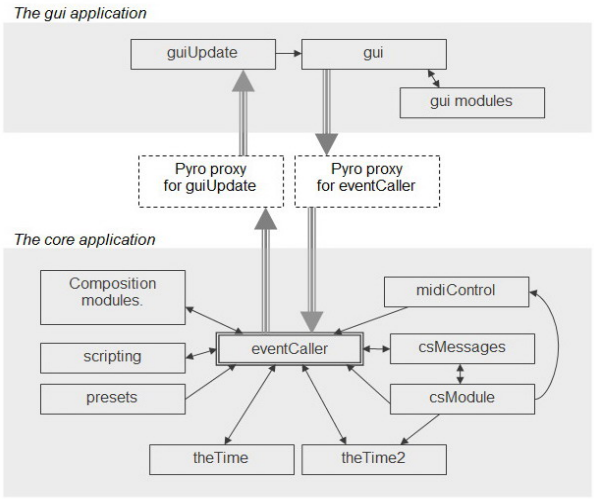

Figure 4: Current ImproSculpt 4 architecture

The two students reported spending much more time in understanding the system and analysing it and planning the changes, than on performing the actual changes; ”As we grew more comfortable with the code, modifications became easier to perform”. After performing packaging of the original files they have iteratively modified many of the files. One of the strategies was to shift pieces of code from one package to another (where it logically belonged). Furthermore, information hiding was improved by imposing proper class structure on some packages (i.e. csMessages). Several pieces of code were rewritten and generalization of some method groups with similar functionality into a single method was introduced. On the last stage of their work the students added comments to the code, in order to facilitate automatic documentation generation.

3.3 Testing

ImproSculpt is being continuously tested by its developer and main user – Brandtsegg. Its functionalities have been intensively tested manually before every official release.

On the other hand, ImproSculpt Classic was used to test the functionality of Csound5 before release, as it was one of the most intricate and complex Csound orchestras to date.

3.4 Process Model and Project Management

The software development model might be classified as agile. No particular agile method was strictly and consciously followed, but rather the agile principles can be observed. The development team was consisting of either the artist only or of the artist and one/several students and in some cases there were external collaborators, thus always a small team. All participants were highly motivated to deliver working product. In the periods when more than one developer was involved there was close collaboration and information exchange. No formal deadlines and milestones were defined. New functionalities were developed and tested constantly in iterations where a stable version of the software was produced for the specific performances or installations. Creation of comprehensive documentation has not been a priority.

The project has started about eight years ago (i.e. in 2000) and since then it has been continuously evolving (i.e. in development phase). The artist predicts at least 2-3 more years of continuous development of the software, or even unpredictably more. Meanwhile Flyndre installation is planned to exist for 10 years (i.e. until 2016), thus the utilization of ImproSculpt should be maintained.

No single sponsor can be identified for ImproSculpt. However, parts of the software were developed for commissioned artworks, including several art installations, music compositions and live performances.

Considering the developers team, Øyvind Brandtsegg is the main developer and owner of ImproSculpt and its variations. The artist Johan Bakken has proposed several variations for ImproSculpt logo and had worked on the web page of Flyndre in collaboration with Geir Arne Brevik. The following people have been working on parts of the ImproSculpt project: Thom Johansen, Torgeir Strand Henriksen, Thor Arne Gald Semb, Audun Smege, Rune Flobakk, Thibault Collet, Maximo Ramirez Robles (all NTNU students with supervisors Sigurd Saue and Letizia Jaccheri).

The csound community as a whole has contributed greatly to the development of ImproSculpt. Snippets of code have been copied from publicly available sources and from discussions on the csound user and developer lists.

3.5 Development Environments and Tools

The current version of ImproSculpt is developed with Csound[3] and Python[4]. It uses also wxPython[5], Numpy[6], Midi[7], psyco[8] and Pyro[9], TickyClav[10]. The documentation standard for

---

[3] Csound - a programming language for sound rendering and signal processing http://www.csounds.com/

[4] Python - a dynamic object-oriented programming language, http://www.python.org/

[5] Python implementation of the C++ library wxWidgets (formerly wxWindows) – a cross-platform GUI toolkit extension for Python, which also provides a handful functions, like timer and thread modules.

[6] An OSS package for Numeric Operations in Python http://sourceforge.net/projects/numpy

[7] MIDI – Musical Instrument Digital Interface – industry-standard electronic communication protocol that defines each musical note… MIDI does not transmit audio – it simply transmits digital information about a music performance.

[8] Psyco, the Python Specializing Compiler, available at http://sourceforge.net/projects/psyco/

[9] Pyro - Python Remote Objects – and OSS Distributed Object Middleware for Python, available at http://sourceforge.net/projects/pyro/

[10] TickyClav - VST instruments used in ImproSculpt - freeware by Big Tick Audio Software. The VST instrument plugins are

ImproSculpt4 is Epytext format, using Epydoc[11] as the autodoc tool. The soundfonts currently loaded by ImproSculpt are all freely available[12] (e.g. Dsix magic, Steinway Grand Piano, etc.). TortoiseCVS has been chosen as CVS client and Putty was used for generation of public/private SSH key needed to access SourceForge and upload ImproSculpt. Furthermore, the information about ImproSculpt, including links to SourceForge, documentation and installation instructions is published in a Wiki. pmWiki was chosen by the students responsible for this task, because it uses files to store the data.

Both Csound and Python code is in itself platform independent, but requires Csound and Python run-time environments installed on the desired platforms. Both of these are available for all popular operating systems, like Windows, Linux/Uniz/BSD, MacOS, Amiga and many others. Python provides interactive interpreter. You can actually give lines of code to the interpreter while a program is running, effectively programming on-the-fly, making debugging and experimentation easier.

In ImproSculpt, Brandtsegg wants to utilize MIDI as a control source for parameters in the system. Csound provides an implementation of the MIDI standard.

Øyvind Brandtsegg has chosen to use Python. Ideally, Brandtsegg wishes to make the software so accessible that other musicians using ImproSculpt can add to and extend it, as it is an open-source project.

The ImproSculpt version in Flyndre uses Python, wxPython, Csound, Doxygen[13], MIDI.

3.6 OSS Publishing

Originally created as a tool for himself, Brandtsegg quickly discovered that his ideas sparked a lot of interest from other composers and music technology enthusiasts, and Brandtsegg decided to declare ImproSculpt as an open-source distribution. Thus, he uploaded the ImproSculpt's source code on the net.

Making the software publicly available was supposed to help the artist, whose main focus is the compositional part, to recruit people to help, to improve, to give ideas from other viewpoints, to find bugs, etc. Although ImproSculpt was declared Open Source Software by Øyvind few years earlier (when the source code has been accessible on internet), a lot of management tasks were required before it was really made Open Source.

Two NTNU students as part of their pre-thesis study [2] were involved in the publishing of ImproSculpt as OSS. At that time the software was available as "ImproSculpt bundle (Flyndra, ImproSculpt Classic and Unstable ImproSculpt)". Students' intervention included licensing of the ImproSculpt source code, adding the appropriate terms and conditions documents in the software packages, publish the software on SourceForge, setting up a CVS environment and a Wiki site.

Currently, ImproSculpt is licensed under the GNU General Public License (GPL) schema. It is uploaded on SourceForge at http://sourceforge.net/projects/improsculpt/. This type of license was chosen mainly due to its popularity. The artist Øyvind Brandtsegg stated "*What I want from the license is to give us protection against theft of the code into commercial closed source projects*". The developers believe that the GPL license will best guarantee such high protection.

SourceForge was chosen because it provides most or even all necessary tools for managing OSS. In addition, it was expected that the publication in SourceForge will be a good "advertisement" of the ImproSculpt, because it will be "visible and accessible to everyone". Meanwhile, the Wikis were supposed to increase the attractiveness of the project for potential developers by offering detailed and well organized information. The students report that at the beginning of their intervention the information related to ImproSculpt was scattered. As their supposition was that the potential developers will be also end-users, a great portion of the effort was planned to be on improving installation documentation, thus that it will be easy for anybody to get started. The two students had planned to establish a backup policy. They have added "credit" file with the names of all contributors to the project.

Publishing ImproSculpt as open source did not increase developers' participation as expected. We speculate that the reason is in the still difficult installation and setup of the whole system. As earlier described ImproSculpt sits on top of Csound and Python and relies on several additional packages. Each software has its own installation and setup instructions which often change with new versions. Compatibility issues sometimes appear too. In our viewpoint, it seems that a lot of artists/programmers want to write their own software, and a lot of artists/non-programmers do not know how to use it. An automated installation procedure should be devised, as this will most probably increase the number of potential users and developers of the system.

4. CHALLENGES & LESSONS LEARNED

In this section we provide a reflection on which were the most significant development challenges and what lessons were learned throughout the development of ImproSculpt.

4.1 Development Challenges

One of the biggest challenges with the development of ImproSculpt has been the fact that the application is a moving target. At times, the specifications change faster than the implementation time. This was especially relevant during the early phases of development. Currently, specification changes are limited to the "local module" scope, and are easier to deal with.

---

only guaranteed to work under windows, however it is relatively easy to disable the loading of the VST plugin and this has only small ramifications on ImproSculpt usability and flexibility. Available online at http://bigtick.pastnotecut.org/index.php?action=PROD&pcode=120

[11] Epydoc - Automatic API Documentation Generation for Python, available at http://epydoc.sourceforge.net/

[12] HammerSound web site - http://www.hammersound.net/

[13] A documentation system (i.e. a tool for generating documentation) for C++, C, Java, Objective-C, Python, IDL (Cobra and Microsoft flavours) and to some extend PHP, C#, and D.

Utilization of multiple CPUs or multi-core CPUs has been found very challenging too. Audio processing is done on a buffer of audio samples, this buffer must be synchronized with the previous and next buffer. This makes multi-CPU processing difficult. Realtime performance is a continuous challenge and can never be good enough. Current solution is to run the different parts of the system as separate applications (allowing audio processing one dedicated CPU/core).

In addition, timing problems appeared and difficulties in designing a stable and precise clock for timed automation. The main problem is that some decisions have to be calculated at the moment when the answer is needed (to take any external changing parameters into account). The calculation does sometimes require time, but we do not want to get the result too late. This conceptual problem can only be solved in a satisfactory way by optimizing the calculations and getting faster computers.

4.2 Lessons Learned

Reflecting back on the development of ImproSculpt we found several issues to be important to share with other developers and researchers.

Throughout the project one can see the participation of several people – NTNU students, other artists and developers. Nevertheless, the artist - Øyvind Brandtsegg - is the initiator, project-leader, software owner, main developer and principle decision maker during the development of the artistic software. The functional requirements for the system were designed and developed by the artist and the software developers were not competent to influence on them. The high-level functional requirements were rather abstract (e.g. live sampling and manipulation) and with quality attributes defined [definable] mostly within the art domain. The specific functional requirements were often changing [growing with time], as new ideas for musical expression appeared on day-to-day basis (i.e. the artist was continuously coming up with new ideas). There was no planning of functional requirements in advance, due to the experimenting nature of the artist. For the same reason there is no formal project schedule, apart from deadlines of commissioned works. The choice of technology (programming languages, standards, etc.) was done by the artist. The artist took care that the selected tools/languages are, as much as possible, platform independent.

The lack of software engineering knowledge started to show-up as the project grew bigger. The lack of clear architecture and more particularly the lack of modular separation led to difficulties in adding new functionality. It also did not facilitate inclusion of other (new) developers. Modularization and re-use of code were very important in ImproSculpt, as they makes it easier to maintain and easier to read. Without the application of software engineering knowledge the errors made in the first software version (even if they had been realized) were probably going to be repeated in the new one.

The collaboration of the artist with the software engineering students was fruitful for both sides. If the project was to be repeated the artist would still collaborate with students. The artist, however, realized that in order have better results students need to be given more specific (i.e. concrete and well defined) tasks.

During the following years after students' intervention the artist has continued his solo work on ImproSculpt and the software has evolved and changed several times. The artist, however, has consulted and reused whenever possible the modular architecture proposed by the software engineers and he found it very useful. In addition, the artist found helpful the publishing of the software as open source in SourceForge and the utilization of the Wiki and the Concurrent Versioning System (CVS) introduced by the software engineers.

OSS was requested/needed because of the desire for wider distribution of the software, help from a bigger community and sharing ideas with interested people. However it was also needed as an intellectual property protection mechanism.

For the artist, who is also the main developer, the flexibility was more important than ease of use. For him it is important that everything is possible, and then limit the complexity for each specific case of using the application. In this way, as little as possible is hidden from him and he has a wide range of possibilities open. Potential users, however, might be scared off by a huge and complex application that can do "everything". Users want small and neat applications that are easy to understand quickly.

5. RELATED WORK

Since their creation computers have been used in art domain [7]. Software applications have been often created for the special needs of artistic projects and as part of the artworks, sometimes being quite exigent in terms of computational power or resources [8]. Software engineering issues in art-related projects have been studied by several researchers in the recent years.

Requirements elicitation has been identified as one of the most difficult issue in multidisciplinary projects comprising software developers and artists [9]. Often problems appear as requirements by the artist might change repeatedly until he/she is satisfied [10]. Defining the system specification in enough tangible terms in early stage of a project has been found particularly challenging [11]. 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.

Considering software architecture, different approaches have been taken depending on the concrete functionalities covered by the application. Nevertheless, modularity has been observed in many reported works, like [12, 13, 9]. The modular approach allows reuse of software components and facilitates the modifiability of the application.

The tools utilized for developing software for artistic purposes varies widely. Some authors [14] discuss the need of tools or environments that do not require programming by the artists. Design of domain specific visual languages (e.g. [15, 16]) is proposed as a valuable approach. Other authors [17], however, argue that programming knowledge is required to allow freedom and flexibility in developing innovative applications by domain experts (e.g. artists).

An extended review of software engineering issues in art domain can be found in separate articles [18, 19].

6. SUMMARY

In this article we present ImproSculpt, the development process, the main challenges and the lessons learned from it.

ImproSculpt is software for live music improvisation and for algorithmic composition based on sampling. Øyvind Brandtsegg is the musician/composer and the main developer. ImproSculpt has been used for the music composition in many performances, recordings and several art installations.

From software engineering perspective the software development of ImproSculpt might be classified as agile. The small team, often only the artist/developer, had no strict formal requirements specification and development planning. Documentation has not been a priority. In fact, software engineering knowledge has appeared to be needed only when the complexity of the software grew significantly. Performance, availability and modifiability have been identified as most important non-functional requirements. Improved modifiability has been reached by adjusting the software architecture and splitting functionalities into modules. This also simplifies the reuse of components.

Several important lessons had been learned. First of all, the development of the software is strongly influenced by the artists' creative and experimentive style - functional requirements often change, as artistic ideas evolve and new ones continuously appear. This makes functional requirements often impossible to define in a formal way for external developers to implement. This issue deserves extended further research from software engineering point of view.

Several NTNU students have participated as software developers throughout the development of ImproSculpt. The artistic project was well accepted and useful for software engineering students. It appears to be inspiring for the student and is good practical work. Students' work has been appreciated also by the artist. Nevertheless, it has been found important to give to students rather well defined tasks.

Finally, ImproSculpt has been published as open source with two main objectives: 1) to protect the intellectual property and 2) to attract more developers and users. We believe that the chosen license (i.e. GPL) deals well with the first objective. To fulfill the second one the most popular open source repository was utilized (i.e. SourceForge). It was, however, not enough to draw new developers into active participation. The complexity of the installation and setup of the system might be the reason, thus further study in this direction is required.

7. REFERENCES

[1] Semb, T. A. G. and A. Småge, "Software Architecture of the Algorithmic Music System ImproSculpt", in *Dep. of Computer and Information Science, Fac. of Information Technology, Mathematics and Electrical Engineering*. vol. Master Trondheim, Norway: Norwegian University of Science and Technology (NTNU), 2006, p. 70.

[2] Collet, T. and M. Ramirez, "IMPRO SCULPT: Open Source - Artistic Software ", Norwegian University of Science and Technology (NTNU), Trondheim, Norway, Report for TDT4705 - Software Engineering, Depth Study, 2006, available online at http://www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2006/When_OS_meets_AS.pdf.

[3] Ratkje, M., "CD: ADVENTURA ANATOMICA", 2005.

[4] North, M., "Web page of Ottawa Proccessed - a three piece set of improvisations", http://www.sonus.ca/curators/thomson/north.html, last visited 14/12/2007.

[5] Brandtstegg, Ø., "Øyvind Brandtstegg web site", http://oeyvind.teks.no, last visited 19/11/07.

[6] Garioud, A., C. Lebegue, G. Mayende, H. M. Gundersen, M. L. Hæreid, and T. Engum, "Flounderphonics", Norwegian University of Science and Technology (NTNU), Trondheim, Norway, Product Report for Expert in Team - Art and Software Village, 2006, available online at http://www.idi.ntnu.no/~letizia/eit2006/reports/ProductReportG2.pdf.

[7] Sedelow, S. Y., "The Computer in the Humanities and Fine Arts", *ACM Computing Surveys (CSUR)*, vol. 2 (2), pp. 89-110, 1970.

[8] Polli, A., "DATAREADER: a tool for art and science collaborations", in *Proceedings of the 12th Annual ACM International Conference on Multimedia* New York, NY, USA: ACM Press, 2004.

[9] Marchese, F. T., "The Making of Trigger and the Agile Engineering of Artist-Scientist Collaboration", in *Proceedings of the Conference on Information Visualization (IV)*, 2006.

[10] Machin, C. H. C., "Digital artworks: bridging the technology gap", in *Proceedings of The 20th Eurographics UK Conference, 2002* 2002, pp. 16-23.

[11] Biswas, A. and J. Singh, "Software Engineering Challenges in New Media Applications", in *Software Engineering Applications (~SEA 2006~)*, Dallas, TX, USA, 2006.

[12] Boyd, J. E., G. Hushlak, and C. J. Jacob, "SwarmArt: interactive art from swarm intelligence", in *Proceedings of the 12th Annual ACM International Conference on Multimedia* New York, NY, USA: ACM Press, 2004.

[13] Fels, S., Y. Kinoshita, C. Tzu-pei Grace, Y. Takama, S. Yohanan, A. Gadd, S. Takahashi, and K. Funahashi, "Swimming across the Pacific: a VR swimming interface", *Computer Graphics and Applications, IEEE*, vol. 25 (1), pp. 24-31, 2005.

[14] Edmonds, E., G. Turner, and L. Candy, "Approaches to interactive art systems", in *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* Singapore: ACM Press, 2004.

[15] Bestor, C., "MAX as an overall control mechanism for multidiscipline installation art", *Computers & Mathematics with Applications*, vol. 32 (1), pp. 11-16, 1996.

[16] Gross, J. B., "Programming for artists: a visual language for expressive lighting design", in *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, 2005, pp. 331-332.

[17] Smith, B. K., "Design and computational flexibility", *Digital Creativity*, vol. 17 (2), pp. 65-72, 2006.

[18] Trifonova, A., S. U. Ahmed, and L. Jaccheri, "SArt: Towards Innovation at the intersection of Software engineering and art", in *Proceedings of The 16th International Conference on Information Systems Development* Galway, Ireland: Springer, 2007.

[19] Trifonova, A., L. Jaccheri, and K. Bergaust, "Software Engineering Issues in Interactive Installation Art", *Inderscience Int. J. of Arts and Technology (IJART)*, vol. 1 (1), 2008.