Masterstudium Computermusik und Klangkunst

Studienkennzahl V 066 705

# Influences of computer music tools on my composition

**WRITTEN PART OF THE ARTISTIC MASTER’S THESIS**

Kunstuniversität Graz

Institut für Elektronische Musik und Akustik

Vorgelegt von: Alisa Kobzar (Matrikelnummer: 11836155)

Wissenschaftlicher Betreuer: Univ.Prof. Dr.phil. Gerhard Eckel

Künstlerischer Betreuer: Mag.art. Mag.phil. Mag.rer.nat. Daniel Mayer

Abgabedatum: 12.01.2026

Alisa Kobzar……………………… 11836155………..
(Name in Blockbuchstaben) (Matrikelnummer)

# Ehrenwörtliche Erklärung

Hiermit bestätige ich, dass mir der *Leitfaden für schriftliche Arbeiten an der KUG* bekannt ist und ich die darin enthaltenen Bestimmungen eingehalten habe. Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen nicht verwendet und die wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Sofern fremde Hilfe (z.B. Korrekturlesen durch Native Speaker) und/oder KI-Dienste verwendet wurden (z.B. internetbasierte Übersetzungstools und/oder KI-basierte Sprachmodelle) habe ich dies ausgewiesen.

[illegible]

## Abstract

This[1] thesis investigates how computer music tools shape compositional thinking, workflow, and artistic outcome. Challenging the longstanding assumption of software neutrality, the research approaches the question through an auto-ethnographic framework, examining how my own practice has transformed as I engaged with a range of creative coding environments, analytical systems, and modular workflows. Six case studies form the core of the inquiry, each demonstrating a distinct mode of technological influence. Post-case reflection traces a shift from assuming control over tools to negotiating with them as co-agents in the compositional process. Using the Cognitive Dimensions framework, the thesis analyses how modular, multi-software workflows distribute thinking across different representational systems. The final chapter addresses the fragility and lifespan of computer music tools, proposing strategies for sustainability and future-proofing computer music works. Taken together, the thesis demonstrates that software does not merely facilitate composition but actively conditions it, influencing aesthetic outcomes and contributing to the formation of compositional identity.

[1] Throughout this thesis, I have used ChatGPT (free version) to help me reformulate my thoughts and maintain consistent language usage.

## Zusammenfassung

Diese Arbeit untersucht, wie Computermusik-Tools das kompositorische Denken, den Arbeitsablauf und das künstlerische Ergebnis beeinflussen. Die Forschung stellt die seit Langem bestehende Annahme der Software-Neutralität in Frage und nähert sich dieser Frage im Rahmen einer autoethnografischen Analyse, indem sie untersucht, wie sich meine eigene Praxis verändert hat, als ich mich mit einer Reihe kreativer Programmierumgebungen, Analysesystemen und modularen Arbeitsabläufen beschäftigt habe. Sechs Fallstudien bilden den Kern der Untersuchung, wobei jede eine bestimmte Art des technologischen Einflusses aufzeigt. Die Reflexion nach den Fallstudien zeichnet eine Verlagerung von der Kontrolle über die Werkzeuge hin zur Verhandlung mit ihnen als Mitwirkende im Kompositionsprozess nach. Unter Verwendung des Cognitive Dimensions-Rahmens analysiert die Masterarbeit, wie modulare, Multi-Software-Workflows das Denken auf verschiedene Darstellungssysteme verteilen. Das letzte Kapitel befasst sich mit der Fragilität und Lebensdauer von Computermusik-Tools und schlägt Strategien zur Nachhaltigkeit und Zukunftssicherheit von Computermusikwerken vor. Insgesamt zeigt die Arbeit, dass Software die Komposition nicht nur erleichtert, sondern aktiv mitgestaltet, ästhetische Ergebnisse beeinflusst und zur Bildung einer kompositorischen Identität beiträgt.[2]

[2] The abstract was translated with the help of the DeepL Translator (https://www.deepl.com/translator, accessed on January 7, 2026), and then checked by me.

## Table of contents

# I. Introduction

How does the choice of software shape the creation of a sound artwork—both in its process and in its outcome? For a long time, I assumed that a composer or sound artist could achieve almost any result with any software belonging to the same paradigm. I trusted in the flexibility and often-claimed limitless affordances of established creative coding environments such as SuperCollider and Max/MSP, believing that the choice of software had little or no impact on the final sound.

Over time, however, I began to question this assumption. Certain compositions unmistakably bear the sonic signature of Max/MSP or SuperCollider. Composers, upon adopting a new environment, often begin to think and work in unfamiliar ways, subtly reshaping their creative processes and aesthetic decisions. These observations led me to reflect critically on how software tools influence compositional thinking and outcomes.

This text presents an auto-ethnographic exploration of how computer music tools have shaped my own compositional practice. Since 2018, I have integrated numerous software environments and algorithms into my work, and through this experience, I aim to examine how specific tools and modular workflows affect my creative decision-making, my conceptual approach to sound, and ultimately the music I produce.

The structure of this thesis moves from framing the problem to examining it in practice, and finally to analysing the broader implications for compositional method. I begin by outlining the motivation for this research and locating it within existing discussions on the influence of digital tools in composition. I then proceed with definitions of what is meant by computer music tools in the context of my work, describe their portraits, as well as clarify their roles within artistic practice.

The core of the thesis consists of six case studies, each examining a different composition and the specific way in which a tool or environment shaped its development. Following the case studies, I reflect on how these experiences reshaped my compositional practice as a whole. Building on this reflection, I apply the Cognitive Dimensions framework to analyse my modular workflow in depth, using my composition *What Is Left* as a focused example.

The thesis then turns to the question of longevity, examining the lifespan of computer music tools and the implications of software and hardware fragility for sustainable compositional practice. Strategies for maintaining performability, ensuring future adaptability, and designing workflows that remain resilient across technological change are considered in relation to my own working methods.

Through this structure, the thesis traces the reciprocal relationship between composer and software, showing how tools not only support creative work but actively shape artistic identity, aesthetic outcomes, and the conceptual foundations of compositional thought.

## II. Motivation

My professional background spans instrumental composition, piano performance, musicology, and teaching. In parallel, I have explored video post-production and experimented with a range of tools for creating audiovisual and interactive compositions. My decision to study computer music and sound art was initially motivated by a desire to expand my compositional toolkit, to understand more deeply how computers participate in the creative process, and to develop skills for producing multimedia art. For many years, however, I did not question the influence of these tools on my artistic decisions across musical, visual, or intermedia practices. I regarded software primarily as a means to an end, rather than as an active factor shaping creative thought.

In this text, I investigate how specific software environments influence compositional thinking and the resulting sound artworks, combining theoretical reflection with practice-based artistic inquiry.

I proceed with evaluating my modular workflow, seeking to understand the motivations and intuitions that guided my framework for combining complex software tools. By systematically studying these influences and my own choices, I aim to cultivate a more conscious and reflective compositional practice, in which the idiomatic affordances and constraints of computer music tools are acknowledged, understood, and purposefully integrated into the creative process.

## III. Literature overview

My awareness of the critical role of software in shaping compositional practice began with the article by J. Snape and G. Born[3], who offer an updated portrait of Max/MSP—a software now deeply integrated into artistic and educational contexts worldwide. Their central argument, which I take as a key insight for this thesis, is that Max/MSP is not a neutral tool, despite its self-presentation in marketing materials. Like any software for computer music, it carries specific assumptions and priorities embedded in its design. J. Snape and G. Born observe that users often adopt patterns and aesthetics that stem directly from the software's built-in tutorials, help files, and example patches. These resources reflect not only technical design but also the ideological imprint of the developers and the company behind them. Consequently, the so-called "neutral" affordances of a software environment are often guided by corporate strategy and user targeting. In the case of Max/MSP, its design has increasingly been directed toward closer integration with Ableton Live, reflecting the commercial interests of Cycling '74 and its parent company. This has broader implications for both aesthetics and sustainability. While open-source developers continue to extend Max/MSP through external libraries, the main software evolves according to the company's upgrade cycle. This leads to a common issue: the instability and obsolescence of user-made patches and libraries, and raises an important question: how can we compose computer music works that remain performable ten years later, despite software evolution and system incompatibilities? And to what extent do our works depend on the software-specific features that may disappear over time?

Similar reflections appear in the work of G. Nierhaus, who emphasises the impact of software choice on the compositional process. He notes that "especially the choice of a certain software and already a specific view on the musical material turns into a decisive factor in, often interlinked, generative and analytical approaches."[4] The software's visual design, its way of representing musical material, and its treatment of time and interaction all influence how a composer

[3] Snape and Born, „Max/MSP, Music Software, and the Mutual Mediation of Aesthetics and Digital Technologies.", p. 221

[4] Nierhaus, „On Composers and Computers", p.1

perceives and develops sound. M. Marrington, mentioning the notion of technological determinism, cites A. R. Brown, who points out that every medium encourages particular kinds of artistic output. Brown warns that “only when we ignore the transforming nature of a medium, or deny it, we risk the transformational change taking us by surprise or undermining our true intention.”[5] T. Magnusson, in *Confessions of a Live Coder*, reflects on his experience adopting a new programming paradigm for musical creation, referencing J. McCartney, the creator of SuperCollider: “each musical programming language or software has its own functional character, and the user learns to think according to its ways.”[6] This observation highlights how software environments not only provide tools but also teach us how to think musically within their frameworks.

These insights are reinforced by A. R. Brown, who argues that the influence of computer music tools on compositional outcomes is mediated more by personal engagement and cultural context than by the tools themselves. Effective computer music composition, he suggests, emerges not from domination over a system, but from collaborative working with the tool, allowing it to shape creative possibilities. Similarly, O. Bertelsen et al. describe software like Max/MSP as analogous to a traditional instrument, emphasising its performability, its distinct sound signature, its inherent capacity for musical exploration, and the disciplined skills required to exploit its potential fully. Their interviews with composers reveal that software becomes not only a means of production but also a subject of artistic inquiry in itself.

Further perspectives come from studies of notation and programming interfaces. C. Nash[7] uses the Cognitive Dimensions of Notation framework to show how software environments—whether a score, a DAW, or Max/MSP—shape the user’s actions by affording or constraining certain kinds of musical operations. By dissecting usability factors such as ease of error detection, visibility of components, and consistency of feedback, Nash demonstrates that compositional behaviour is deeply intertwined with the cognitive ergonomics of the tools used.

[5] Brown, „ Computers in Music Education: Amplifying Musicality“, p.9 in: Marrington, “Paradigms of Music Software Interface Design and Musical Creativity”, p. 3

[6] Magnusson, „Confessions of a Live Coder“, p.1

[7] Nash, „The Cognitive Dimensions of Music Notation“, p. 2

Finally, O. Green et al. examine creative-coding environments such as Max/MSP, Pure Data, and SuperCollider, highlighting both their shared focus on real-time interaction and their divergent architectures. While Max/MSP and Pure Data emphasise a visual, process-oriented patching idiom with limited data manipulation facilities, SuperCollider's textual, client-server architecture offers sophisticated timing and patterning capabilities, supporting complex analyses and musical structures. The study underscores that each environment enforces distinct compositional constraints and enables unique artistic strategies, shaping the composer's approach to sound and interaction.

A. McPherson and K. Tahıroğlu[8] discuss the idiomatic patterns of each programming language, comparing them to instrumental music figures within a specific style. They survey popular computer music software and its non-neutrality. However, they don't explain the cause-and-effect relationship between language design patterns and the resulting instruments and compositions. Instead, they suggest that composer and language interact in a complex way. From the perspective of digital music instrument creation, they highlight the inherent models in each language and the invisible connection between language creators and instrument designers. Ultimately, they emphasise the presence of aesthetic influences on different languages.

I find this process of self-observation and reflection essential for any composer working with computers. It reveals the often-hidden influences of software on compositional thought and helps us understand the transformative nature of our tools. The theoretical frameworks described above provide a lens through which I can observe my evolving practice.

## IV. Computer music tools

In this research, I understand computer music tools broadly, encompassing both hardware and software that facilitate artistic engagement with the computer. These tools allow the computer to function in multiple roles: as a tool, an instrument, a medium, a collaborator, and an environment. Rather than a fixed set of objects, they form a fluid ecosystem that I continuously rethink and reassemble

[8] Tahıroğlu and McPherson, „Idiomatic Patterns and Aesthetic Influence in Computer Music Languages.", p.54

for each composition. This ecosystem can include software and creative coding environments such as Max/MSP, SuperCollider, Pure Data; algorithms or plugins, for instance spectral analysis tools or pitch-tracking modules; hardware interfaces including controllers and sensors; compositional environments like notation software or digital audio workstations; different programming paradigms ranging from text-based to visual programming; and, increasingly, AI systems or generative models.

The way a composer interacts with these tools shapes the computer's role within an artistic context. A. R. Brown describes the computer as a tool, instrument, and medium in the context of music education. As a tool, the computer acts as an extension of the composer's hands and mind, offering efficiency, precision, and automation for tasks such as notation, synthesis, or editing.

When considered as an instrument, the computer becomes a performative object capable of expressive interaction. In this role, the composer or performer learns the system's tactile[9] and responsive qualities through live coding, gestural control, or real-time processing.

As a medium, the computer structures how music can exist and be perceived. Its underlying algorithms, data structures, and interfaces define the expressive possibilities and aesthetic constraints of the work, influencing both its form and its sonic characteristics.

Computer and computer music tools can also act as collaborators[10], contributing agency and unpredictability to the creative process. Systems based on AI, machine learning, or generative algorithms can generate ideas, offer resistance, or make autonomous decisions, requiring the composer to negotiate authorship with the machine.

Finally, computer music tools constitute an environment[11]—a space of artistic inhabitation that encompasses all other roles. This environment is a living

[9] In this text, I understand "tactility" as a system's ability to encourage users to easily delve into its code and implementation behind the front end or UI to adapt the system to their creative needs.

[10] Tipei, "The Computer: A Composer's Collaborator.", p. 190

[11] Negrete-Yankelevich and Morales-Zaragoza. *The Apprentice Framework: Planning and Assessing Creativity,* p. 3

ecosystem of scripts, workflows, interfaces, and habits in which the composer develops long-term relationships with technology.

## V. Modes of compositional engagement

A. R. Brown[12] identifies four primary modes through which a composer can engage with musical material using technology: observer, director, player, and explorer. These modes describe the relationship between the composer and the medium, revealing how ideas interact with the system's capabilities and constraints.

In the observer mode, the composer treats the work as a complete artefact, critically evaluating it from a detached perspective. The director mode involves actively shaping musical material, using the software as a tool that responds to the composer's instructions. In the player mode, the composer engages with the medium as an instrument, interacting intuitively and expressively through a personal, tactile relationship. The explorer mode emphasises experimentation and discovery, navigating the conceptual or technological space of the system to uncover new musical possibilities.

## VI. Expanding my software ecology

Before engaging with SuperCollider, my experience primarily involved digital audio workstations (DAWs) and graphical patching environments, each supporting distinct modes of musical thinking and engagement. Working within a DAW framed my composition through linear time, arranging materials on a timeline much like writing a score. This environment emphasised planning, structure, and control, fostering the director and observer modes, where I thought from above and organised material into seemingly complete forms.

Graphical patching environments, such as Max/MSP, invited a different mindset. I became a builder and performer, designing instruments and systems for real-time interaction.

Transitioning to text-based environments, particularly SuperCollider, transformed my relationship with composition. Moving from visual to symbolic thinking allowed me to focus on processes, systems, and generative behaviours

[12] Brown, „Modes of Compositional Engagement.“, p. 8-16

rather than static objects or timelines. Writing code felt cleaner and more navigable—there were no tangled cables, and structures could be easily modified. Composition became an exercise in designing relationships and behaviours rather than arranging discrete sounds. In this shift, my dominant modes of engagement also changed. Max/MSP had naturally supported player and explorer modes, encouraging live experimentation and discovery, while SuperCollider fostered director and observer modes, promoting abstraction, algorithmic thinking, and reflective decision-making.

Over time, I explored a variety of software systems—including OpenMusic, Max/MSP, SuperCollider, Pure Data, and several DAWs—not only for practical purposes but also to find environments aligned with my artistic thinking. During my studies at the Kyiv National Music Academy of Ukraine (2009-2014), I used mainly OpenMusic and Max/MSP for computer-assisted composition.

Alongside these environments, I studied programming languages including C, C#, Python, and HTML/CSS to understand the logic underlying software and to build custom extensions when needed.

Selecting SuperCollider as my main creative coding environment since 2018 represents a fundamental shift in my compositional practice, both in terms of thinking and workflow. Unlike graphical environments such as Max/MSP, SuperCollider[13] operates as a text-based, client-server system, which separates the act of coding from real-time sound production.

SuperCollider also embodies inherent biases that influence my work. Its architecture and idioms encourage certain ways of structuring sound, approaching time, and designing interactions. For example, pattern-based sequencing naturally leads to thinking in terms of rhythmic cycles, probability, and generative processes, while the client-server model foregrounds the distinction between musical intention and its real-time realisation.

## VII. Affordances, constraints, and the target consumer

Each software embodies a target user, a creative paradigm, and a genre orientation, reflecting the aesthetic values, preferences, and biases of both its developers and intended audience. These layered influences—personal,

[13] McCartney, “Rethinking the Computer Music Language: SuperCollider.”, p. 64

corporate, and cultural—define what is easy or intuitive in a given environment (its affordances) and what is cumbersome or restricted (its constraints). One way to reveal these differences is to attempt the same compositional task across multiple environments. For instance, examples from Andy Farnell's *Designing Sound* were originally written in Pure Data and later adapted to SuperCollider. Even in this adaptation, the distinct environments dictate varying approaches, which are briefly illustrated in the comments to the open-source SuperCollider patches[14] .

Similarly, the FluCoMa[15] project illustrates these subtle differences in practice. During a workshop at the IEM in 2025, Pierre Alexandre Tremblay (one of the FluCoMa developers) demonstrated using the same FluCoMa library in both Max/MSP and SuperCollider. Switching between the two revealed that, beyond implementation details, each environment fosters distinct modes of interaction and conceptual engagement with sound.

A. McPherson and K. Tahıroğlu[16] describe this phenomenon as "idiomatic patterns" in programming languages, analogous to instrumental idioms in music.

## VII.1. The contemporary composer as target consumer

If the previous section examined how software constrains creation, this one considers how the composer themselves becomes a target within the broader technological and cultural economy.

Today, composers face a multitude of overlapping "competitions" among software environments: notation tools, DAWs, computer music languages, and creative coding platforms. Each competes not only in technical quality but also in accessibility, economy (commercial vs open source), and community support.

In this landscape, where multimedia and transmedia practices are increasingly common, composers are often expected to embody a "Renaissance" skill set — composing, programming, designing interactions, building instruments, performing, and conducting research all at once. The culture of contemporary

[14] https://en.wikibooks.org/wiki/Designing_Sound_in_SuperCollider . Accessed January 7, 2026

[15] https://www.flucoma.org/ . Accessed January 7, 2026

[16] McPherson and Tahıroğlu, „Idiomatic Patterns and Aesthetic Influence in Computer Music Languages", p.54

creation demands that artists learn quickly, remain flexible, and stay “original” and “new.”

We are constantly pushed to master the next tool, to adapt to the next update, to navigate between tutorials, YouTube guides, and now AI assistants like ChatGPT — each bringing its own cultural and algorithmic biases. Copy-paste practices, code reuse, and community-driven learning create a shared aesthetic vocabulary, often blurring the boundaries between individual expression and the software’s defaults.

In this context, the choice of software becomes not only an artistic decision but also a strategic one:

- How to invest time in learning a tool that will remain relevant for years?
- How to balance complexity with clarity — to create enough structure for meaningful self-expression within systems that could otherwise feel infinite?
- How to remain flexible across environments while preserving one’s artistic identity?

Learning at least one programming language or creative coding environment — whether text-based (like SuperCollider) or visual (like Pure Data) — provides access to a deeper understanding. Working across both paradigms helps bridge the gap between visual and textual thinking, between immediate feedback and abstract control, between design and composition.

## VIII. Comparing creative coding environments: SuperCollider, Max/MSP, Pure Data

Compositions often “sound” as if they were created in a particular software environment because these environments are cultural–technical systems. They embody assumptions about how musical thought should unfold and how sonic materials are organised. The comparative features of Max/MSP, Pure Data, and SuperCollider—summarised by A. McPherson and K. Tahıroğlu[17] —illustrate how each platform’s core paradigm, interaction model, and affordances inscribe

[17] McPherson and Tahıroğlu, „Idiomatic Patterns and Aesthetic Influence in Computer Music Languages“, p.54

specific ways of listening, composing, and learning, leaving structural and sonic traces in the resulting works.

SuperCollider’s text-based environment and dynamic creation of synthesis processes support flexible, modular structures and rapid experimentation. Its abstraction mechanisms (e.g., SynthDefs and classes) enable users to build higher-level sonic units from low-level components. The lack of graphical elements or presets demands conceptual clarity, positioning the act of composition as the formalisation of ideas in code. Even the help files, often terse and academic, reinforce abstraction as an aesthetic ideal. Consequently, SuperCollider often exhibits evolving textures and processes where listeners perceive the unfolding of algorithms rather than interface manipulation. This environment privileges composers fluent in symbolic reasoning, while beginners may produce simpler works where process dominates over sonic refinement.

By contrast, Max/MSP and Pure Data operate in a visual dataflow paradigm, representing sound as signal paths rather than textual hierarchies, but each carries a distinct ethos. Max/MSP, with its polished commercial interface, fosters interactive and performative compositional approaches. Its visual paradigm encourages a tactile and exploratory engagement with sound, and its help files and presets provide immediate sonic gratification, implicitly teaching looping, layering, and audiovisual integration. The vibrant online community, extensive patch repositories, and copy–paste culture further consolidate a recognisable “Max/MSP sound”. While beginners achieve results quickly, experienced users often combine visual patching with textual coding or external processes to expand expressive possibilities.

Pure Data, though visually similar, embodies a more ascetic, open-source philosophy. Its minimal interface and modular structure encourage a DIY approach, emphasising experimentation over polish. Composers are often required to construct systems from scratch, resulting in a raw, transparent aesthetic. Pure Data works tend to foreground infrastructural sound—oscillators, filters, and feedback networks—over formal finish. Its minimalism and accessibility support hybrid practices bridging software, hardware, and physical computing, giving rise to a “Pure Data sound”.

Comparing the three platforms across dimensions such as paradigm, abstraction, temporal modelling, and interaction reveals how each conditions temporal sensibility and musical form. SuperCollider's real-time creation and destruction of synth nodes supports fluid, evolving architectures, while Max/MSP and Pure Data's static patch graphs favour loops or steady-state textures[18]. Abstraction levels further shape compositional strategies: SuperCollider's class hierarchy enables scalable, modular systems; Max/MSP and Pure Data's subpatch logic encourages local experimentation and performative manipulation. Even the design of core objects (e.g., Pulse and LFPulse in SuperCollider vs. saw~ in Max/MSP vs. [phasor~] in Pure Data) can subtly influence sound manipulation, interaction, and temporal resolution[19].

User communities, shared examples, and pedagogical resources contribute to idiomatic practices associated with each environment, shaping expectations and working methods and could have an effect on the sonic outcome. The "SuperCollider sound" reflects a programming culture valuing generativity and abstraction; the "Max/MSP sound" embodies immediacy, performance, and visual fluency; the "Pure Data sound" arises from experimental modularity and DIY ingenuity. In other words, the audible character of each environment encodes its epistemology—the interaction metaphors, pedagogical assumptions, and cultural values embedded in the software.

As A. McPherson and K. Tahıroğlu note, programming languages carry idiomatic patterns analogous to instrumental idioms: they do not dictate outcomes directly, but they shape habits, expectations, and reasoning. In practice, composers often adapt their ideas to what the software makes easiest, modifying or abandoning concepts that do not align with its workflow. Help files, example patches, and pre-built libraries also leave traces of their creators' aesthetic priorities, subtly guiding the artist's decisions.

Observing these dynamics allowed me to recognise how my own work is conditioned by the tools I choose and use. In the following section, I reflect on my personal experience with SuperCollider, connecting these broader theoretical

[18] McPherson and Tahıroğlu, „Idiomatic Patterns and Aesthetic Influence in Computer Music Languages", p.57

[19] McPherson and Tahıroğlu, „Idiomatic Patterns", p.56

insights about software affordances and compositional agency to the practical realities of my daily creative practice.

## IX. My compositional process

My workflow for composition that combines electronics and instrumental or electronic performers has evolved into a highly iterative and interconnected practice. It begins with conceptual research—defining the topic, interaction model, or main idea—and proceeds through contextual study, technical reading, and material collection, where I record and organise instrumental and electronic sounds into a personal database. I continue with instrumental research to maintain a physical understanding of performance, then prototype in software, experimenting with libraries, code snippets, and tutorials. These prototypes are integrated with live instruments, leading to preliminary mock-ups, detailed composition, and preparation for rehearsals, before testing interactive techniques with performers and finalising the work.

Throughout this process, I engage closely with SuperCollider—or other creative coding environments—maintaining a continuous dialogue between programming and composition. While its community is smaller and documentation less extensive than Max/MSP's, SuperCollider offers liberating flexibility: I can reuse and adapt my own modules, creating a personal library that accelerates the creative phase. A practical challenge remains visual feedback: coming from a notation-based background, I often rely on seeing relationships between musical events. To address this, I use verbose textual feedback, virtual MIDI and audio routing between SuperCollider, Max/MSP, and a DAW, and export files to notation software. Even when finalising scores, SuperCollider operates as the generative and conceptual engine behind the music.

Over time, this workflow has reshaped my understanding of composition. What began as a technical necessity has become a conceptual approach, where systems, processes, and interactions guide the emergence of musical form and behaviour. The act of composing is inseparable from the act of coding, and the constraints and affordances of the software shape both the process and the outcome.

## IX.1. Paradigm shift and changing focus

Adopting a new digital environment does more than change a composer's tools[20]; it introduces a new epistemic and perceptual framework. Working within it means internalising these metaphors—what counts as a "signal," an "object," a "gesture," or a "process." In this sense, software functions as a cognitive technology, not only extending abilities but restructuring thought itself.

Moreover, each software carries historical, cultural, and aesthetic lineages. Learning a new system, therefore, entails a cultural apprenticeship, altering the composer's perception of authorship, complexity, and creative agency.

At a practical level, the environment also restructures workflow and temporality. Timing models, event structures, and abstraction hierarchies influence macro- and micro-level musical form: patches may favour gestural or installation-based works, scripts encourage systemic evolution and generative structures. New tools introduce productive friction—the challenge of translating aesthetic intuition into the software's grammar—which fosters reflection on the boundaries between idea and implementation. In this way, each software functions as a mirror, revealing alternate versions of compositional thinking and creativity.

## IX.2. Transformations in my practice

Working with SuperCollider after Max/MSP has profoundly shifted my compositional habits and priorities. My focus has moved from notation to improvisation, from micromanaging details to designing systems that generate outcomes. Debugging became an integral part of the creative process, and algorithmic composition—loops, patterns, and sequencers—introduced generativity and indeterminacy.

I learned to lower the threshold for evaluating sound alone, understanding that meaning could emerge from structure, behaviour, and interaction. This transformation also demanded balancing technical challenge with creative flow, echoing M. Csikszentmihalyi's notion that creativity thrives when skills match challenges.

[20] Magnusson, "Confessions of a Live Coder.", p. 4

Over time, I developed highly complex projects with SuperCollider in the centre, among which are: interactive systems with metacontrol architectures, intricate mappings of multiple interconnected controllers, OSC-based links to visual and motion-tracking systems within gamified environments to control the spatialization frameworks during a multimedia dance performance, portable systems for computer music processing and battery-powered sound installations using Raspberry Pi, etc. Integrating machine-learning tools, hardware synthesisers, and electrically conductive materials, I constructed interdependent networks where composition and system design overlapped. Yet the lack of visual feedback required constant reflection and improvisation within the text-based environment. I began designing instruments and algorithms intuitively, allowing the system itself to suggest directions and generate surprise, and then analysing and refining these emergent behaviours. This interplay between explorer and player modes of engagement resulted in the application of the metacontrol[21] not only as a part of a collaborative multimedia performance but as a personal compositional language.

In this way, my compositional process became a continuous negotiation between personal intention, systemic affordances, and emergent musical behaviour—a dynamic relationship that naturally leads into examining individual works and case studies.

## X. Case studies

As T. Magnusson asks, “But how would one study the influence of the tool or programming language upon the musician?”[22] I approach this question through an auto-ethnographic lens on practical examples, reflecting on how my ideas and compositional language have adapted to the tools I have acquired over the past years.

In the following case studies, I will analyse six compositions, focusing on different types of tool influence. For some works, the impact of the software was not initially central to my awareness, and here I reflect in broad strokes on process, aesthetics, and conceptual decisions. For other works, the influence of

[21] De Campo, “Loose Control, Gain Influence.”, p. 220

[22] Magnusson, „Confessions of a Live Coder“, p.1

the tools became the primary axis of self-observation, and these are treated in greater detail, with attention to code, patch structures, and hardware interactions.

Each case study begins by defining what counts as a relevant computer music tool for the work at hand (in subtitle). This may include a creative coding environment (Max/MSP, SuperCollider, Pure Data), an algorithm or algorithmic process (pitch tracking, spectral analysis, corpus similarity, machine learning), a hardware interface or sensor (Leap Motion, Myo, accelerometers), a compositional environment (notation software, DAW), a coding paradigm (text-based vs. visual programming), or an AI system (voice models, LLMs). This clarification allows me to identify which technological actor exerted the most significant influence on the piece.

Rather than applying a fixed template across all works, I selected only the questions that were relevant to each composition. In some cases, the tool's impact on form, gesture, or temporal logic was more salient than its sonic defaults; in others, the socio-technical or cultural context—such as corporate AI versus open-source ethos—shaped aesthetic outcomes more strongly. This selective approach ensures that each study remains specific and meaningful, avoiding forced uniformity across diverse projects.

Methodologically, the analysis relies primarily on reflective writing. I reconstruct compositional processes from memory, sketches, rehearsal notes, early audio drafts, and code or patch snapshots. These reflections are complemented by light empirical observation: comparing early and final versions, examining annotated materials, or identifying fragments of code where "tool thinking" manifests. The core remains a self-analytic exploration: how I made decisions, how the tool guided or constrained me, and how the environment shaped my workflow and aesthetic judgments.

From these reflections, I focus on three overarching dimensions: the influence on process—how the tool structured the workflow, sequencing, and improvisational possibilities; the influence on material and aesthetics—how default behaviours, limitations, or biases shaped sonic and structural tendencies; and the influence on conceptual thinking—how the environment reframed notions of authorship, collaboration, control, gesture, and compositional metaphor. When relevant, I also consider the socio-technical context of the selected tool, including its user communities and underlying philosophies.

This modular, reflective method allows each case study to remain grounded in artistic practice while revealing patterns across tools. By examining how my sense of control, authorship, and musical identity shifted with different technological environments, the studies illuminate the subtle but decisive ways in which computer music tools shape both thought and sound. With these principles established, I now turn to the individual works, beginning with the compositions where tool influence became most pronounced.

## X.1. Timbral Approximations / Projections

### Audio-transcription algorithms as compositional agents

In *Timbral Approximations* (for 3 robotic piano players and computer music performer) and *Projections* (for large instrumental ensemble, robotic piano player and computer music performer), I investigate how audio-transcription algorithms—and the errors, limitations, and biases embedded within them—can operate not merely as analytical utilities but as active agents within a compositional process. These works marked a turning point in my practice: instead of treating computational misinterpretation as something to correct, I began to understand it as a source of material, structure, and aesthetic identity[23].

Throughout the project, I worked across several environments, each influencing the workflow in distinct ways. In Max/MSP, I used the Ears[24] library for audio transcription; in Spear, I analysed partials and exported SDIF files; Max/Cage converted these SDIF partials into MIDI piano rolls; MaxOrch[25] (Orchidea) was used both with the open-source library and a custom piano sample dictionary; and in Pure Data, I worked with the halftone analyser patch developed by T. Musil for P. Ablinger's work. The outputs from these systems resulted in symbolic representations—either MIDI or MusicXML—that required pre-processing and quantisation before export. Once imported into Sibelius, the material needed manual refinement. MusicXML retained structural information more accurately but

[23] Kobzar, „Timbral approximation“, p. 2

[24] https://www.bachproject.net/ears/ . Accessed on January 7, 2026

[25] https://perbloland.com/maxorch-about . Accessed on January 7, 2026

frequently produced duration overflows, while MIDI forced destructive assumptions due to the format’s lack of time signatures, tuplets, and bar information. Rather than treating these inconsistencies as external obstacles, I allowed them to enter the creative field.

A central artistic problem emerged early in the process: how to reduce complex instrumental timbre—such as the lowest A of the piano—into the restricted bandwidth of another piano. Instead of working with partials or continuous spectral resolution, the transcription was constrained to the eighty-eight discrete pitches of the keyboard. This limitation produced what I began to think of as a software-constructed pianist, whose logic was defined not by performance practice but by algorithmic rounding. The constraint shaped the compositional framework from the outset: algorithmic bias became an intentional parameter rather than a deficiency to eliminate.

My primary questions concerned the specific biases introduced by these tools and how they influenced instrumental writing involving a computer-music performer. Piano timbre, inherently percussive and unable to sustain or morph sound in the way many acoustic instruments can, interprets other timbres through its own limitations. Conversely, when the ensemble attempted to imitate the robotic piano, the result was not convergence but friction: the ensemble appeared to negotiate with the logic of a machine. This dynamic subtly echoed the concerto principle—not as soloist versus ensemble, but as negotiation between human and algorithmic agents.

The technical configuration reinforced this relationship. The computer acted as an intermediary between a controller (Behringer X-Touch Compact controller) and the robotic piano. The workflow unfolded in three phases. First, I composed the primary musical material. Second, this material was subjected to transcription, which algorithmically “rounded” it into symbolic form. Finally, during performance, the computer-music performer manipulated velocity thresholds and selectively filtered or permitted actions from the controller. This created a layered authorship in which neither human nor system held complete control.

Because this was my first large-ensemble piece after a long break, I had to reactivate my instrumental writing knowledge while adapting to computational thinking. The integration was challenging, especially as I was simultaneously discovering the aesthetic potential of noisy or unstable data. I recalled earlier work with video and motion-capture systems, where jitter and skeletal-solving errors became expressive material. Similar borderline data—generated at the edges of analytical reliability—proved compelling in the sonic domain as well.

Before assessing the influence of these tools, I had to clarify what I meant by "tool." In this context, tools included algorithms, plugins, spectral and pitch-tracking modules, analysis environments, and systems that convert analysis data into symbolic notation. Their influence manifested across four interconnected levels: process, material, aesthetics, and conceptual thinking.

On the level of process, the tools reshaped my compositional strategy. My initial intention was to simulate the impression of near-live transcription of three instruments into piano. Very quickly, the limits became clear: true real-time analysis was not feasible, and the delay between source and transcription was too long to be musically plausible. As a result, I adopted a quasi-live approach using prerecorded material, which was then shaped in real time during performance. This shifted the dialogue structure from simultaneity to call-and-response. The technical impossibility of real-time processing, therefore, directly influenced the form.

The workflow moved increasingly toward a top-down model. The transcriptions frequently produced dense verticalities and chordal accumulations, which led me to conceive the instrumental writing in similar terms—layered textures, repeated notes, and recurring pitch hierarchies. The symbolic visualisations made recurrence rates highly visible, encouraging structural thinking about sound beyond the initial attack as partials decay. At times, compositional decisions were dictated entirely by tool affordances; I was not composing so that the tool would function, but so that it would produce the approximation I wanted.

The influence on material and aesthetics was equally pronounced. The robotic piano's identity emerged from computational behaviour rather than intentional design: granulation, looping, dense chord aggregates, and noisy partial transcriptions. I explored unconventional FFT parameter combinations and distorted analysis settings—not to improve accuracy but to generate musically meaningful divergence. For the first time in my instrumental work, computational error functioned explicitly as material rather than interference, extending earlier visual explorations of error, jitter, and breakdown.

Equally significant were the limitations of what the tools could not provide. Real-time transcription remained unattainable, and no single unified symbolic representation across environments emerged. These constraints shaped the artistic identity of the project as much as its affordances.

Conceptually, the work transformed my thinking about form, time, and morphology. Transcription ceased to function as representation and became a generator of texture. Each attack gained structural relevance due to the piano's percussive nature, which imposed its own behaviour on the material. The guiding question shifted from accuracy—"how closely can this timbre be represented?"—to emergence—"what textures and behaviours arise through transcription?"

From a reflective standpoint, several criteria became central. There was no immediate feedback; all analysis was offline, requiring waiting periods. Parameter choices directly affected complexity and variability. The agency partially shifted to the algorithms, which shaped micro-details inaccessible through manual notation alone. The tools imposed recurring characteristics—especially structurally imperfect or noisy transcriptions—that became embedded in the musical language.

These influences are especially apparent in the robotic piano solo passages, characterised by granulation and looping. My own piano improvisation developed through listening to this material and interacting with it, eventually framing the fragmented transcriptions dramaturgically. Repeated notes with slight pitch deviations and registral arpeggiations emerged from noisy piano transcriptions and later informed the ensemble writing as downward-moving verticals. Transcriptions of other instruments—sometimes clean, sometimes textured, sometimes unstable—entered a pianistic medium and shaped the ensemble's imitative strategies in the vertical passages after the cadenza.

Eventually, the relationship between composer and tool operated as a dynamic of control, negotiation, and resistance. The tools did not simply assist the process—they altered it, shaping not only the material but also the way I listened, perceived, and understood form.

## X.2. PRISM

### Software migration, hardware obsolescence, and the reconfiguration of interaction

In *PRISM* (for instrumental ensemble and live electronics with motion controllers), the evolution of the piece was shaped not by aesthetic intention alone but by shifting technological constraints, discontinued hardware, and a complete transition of the electronic component from Max/MSP to SuperCollider. The work became a clear example of how software and hardware function as structural forces: they altered my compositional thinking, the performance practice, and at times even the identity of the piece itself. Each technological shift required artistic adaptation, and *PRISM* changed accordingly—sometimes minimally, sometimes fundamentally.

The earliest versions were built in Max/MSP and relied on motion-based controllers, particularly Leap Motion and the Myo armband. These interfaces formed the expressive and gestural foundation of the work. When the open-source libraries supporting them became unstable or incompatible with newer operating systems, and when the Myo became obsolete, the interaction model collapsed. The system could no longer sense the same gestures; the performer's learned physical vocabulary disappeared; latency behaviour and failure modes changed. Because the dramaturgy and electronics had been closely tied to the idiosyncrasies of these devices, the disappearance of the hardware effectively removed the original "instrument" of *PRISM*.

This instability became the catalyst for learning SuperCollider. I needed an environment that offered lower-level access, flexibility, and a level of stability that would allow me to maintain or re-implement functionality when necessary. The learning process resembled what Thor Magnusson describes in „Confessions of a Live Coder" [26]: long phases of reading tutorials, borrowing fragments from help files and online discussions, and gradually assembling a system I could understand and extend. Moving into a text-based creative coding environment reshaped my compositional language. Max/MSP encouraged spatial and process-oriented thinking through visual patching, but as the system grew more complex, it also became harder to navigate. In SuperCollider, working directly with synthesis,

[26] Magnusson, "Confessions of a Live Coder.", p. 4

buffers, and algorithmic processes shifted my focus toward timbre, spectral behaviour, and procedural development.

Transferring *PRISM* from Max/MSP to SuperCollider clarified the mapping between performer, controller, and electronics. The system became lighter and easier to maintain, reducing rehearsal stress and increasing confidence for both the performer and me. The interaction logic became more transparent, and with the reduced system complexity, the composition gained clarity. The balance between instrumental and electronic sound became more controlled, and I introduced new processing approaches—such as convolution—that had not been possible in the earlier setup. The most recent version (2025 / #4) builds on this modularity: software, hardware, and sensors function flexibly, supporting cooperative performance and treating the electronics as an integrated instrument rather than a reactive background layer.

The transition also changed how I handled amplitude, granulation, and buffer navigation. Once the mappings became more readable, inconsistencies in level across different sound files became apparent, leading me to normalise and compress the corpus. I removed the second computer that had been required for Leap Motion tracking and began performing part of the electronics myself. This reduced the performer's technical load and increased shared control. Choosing to work with only one controller was shaped by practical constraints—limited rehearsal time made dual-controller interaction unrealistic, but this simplification ultimately supported greater focus and fluency.

The shift in software produced a noticeable aesthetic change. In Max/MSP, the sound world was heterogeneous and somewhat chaotic, reflecting the complexity of the patch structure. In SuperCollider, the textures became more stable and more finely shaped. Granulation developed new characteristics, amplitude behaviour became easier to manage, and the piece as a whole moved toward clarity. Some elements were lost in the transition: SuperCollider did not easily support the visual hand-tracking display used in the earlier version, so I shifted toward working with number streams and abstract data, which proved musically productive in a different way.

Another major transformation occurred when hardware obsolescence removed the original gestural vocabulary. The Myo had enabled pose-based binary gestures; Leap Motion had provided continuous mid-air micro-gestures. When both became unsupported, the embodied identity of the piece disappeared. My performer expressed sadness at losing the Myo, whose physical presence had become part of her role. The only viable replacement was the accelerometer of an Android phone for the *PRISM* version #4. Although I attempted to preserve the continuity of interaction, the movement quality changed as well as expressive potential, and the new device required shared control during performance with a computer musician to maintain expressive range.

These developments raised fundamental questions about the identity of technologically mediated works:

- Is *PRISM* defined by its interaction model or by its sound?
- Does the piece remain the same when the instrument changes?
- Can a gesture–sound relationship survive hardware transitions?

I explored several possible strategies. Re-implementing the original interaction model for every new device was theoretically possible but practically unsustainable. A hardware-agnostic approach—mapping gestural abstractions such as rotation, velocity, or sudden extension instead of device-specific triggers—offered greater stability. Ultimately, I began moving toward an open-hardware approach based on community-supported sensors and OSC-based controllers, reducing dependence on commercial products and their life cycles.

Throughout this process, software and hardware shaped authorship, control, and form. Learning SuperCollider increased shared performance responsibility and allowed me to return to a more musical role rather than maintaining fragile systems. Hardware disappearance required re-examining what constitutes the "instrument" of the piece. The sound world of *PRISM* became more stable and structurally coherent, while still retaining traces of previous versions.

In the end, the work shows how technological literacy, tool stability, and sensing devices influence the development of a composition. *PRISM* remains the same piece, but each iteration reveals a different balance between performer, instrument, and system. The tools do not merely support the music—they reshape it.

## X.3. Floating Pointers

### Hardware sensors and interaction within technology mediation

*Floating Pointers* (versions A–D) for dancer, computer music performer, live electronics, gestural controllers, and video were developed by the duo rotkäppchen between 2021 and 2023. The duo investigates in their various projects how hardware sensors—Leap Motion, Myo armband, OptiTrack, and others —function as digital instruments within a hybrid human-computer and human–human interaction environment[27]. These devices shaped sonic and visual output, dramaturgy, the performer–composer collaboration, and the notion of authorship. The compositional process in *Floating Pointers* unfolded through negotiation of sensor affordances and limitations, mediated by the relationship among dancer, technology, and my musical intention.

Although the hardware remained constant across all four versions, each explored a distinct mapping strategy. In some cases, complex input produced relatively simple output; in others, minimal input generated dense audiovisual responses. This spectrum—from hardware-driven minimalism to software-driven complexity—required continuous mediation between modular software solutions and practical performance constraints.

The live environment emphasised system vulnerability. The interaction between dancer and sensor was often fragile, making failure a dramaturgical element. For example, multidimensional hand postures that produced interesting sonic results in my Leap Motion experiments were impractical for the dancer due to physical effort and movement vocabulary. I simplified the interaction, redesigning the sound space so that the intended sonic outcomes could be reliably accessed in performance.

This led to a collaborative model of *metacontrol*[28], where the digital instrument's control was shared between the dancer, the sensor system, and me. Co-performance and distributed control became standard, and the interaction complexity was limited to match the dancer's movement capacities. Backup strategies—including secondary mappings, override layers, and emergency channels—ensured continuity in case of sensor failure. The traditional

[27] Kobzar, „Sound Object", p. 16-27

[28] De Campo, "Loose Control, Gain Influence.", p. 220

compositional timeline was replaced by interaction vocabularies and action–reaction plans, making dramaturgy central while score-like precision became secondary.

This approach reshaped my compositional method. I do not compose *for* the shared instrument in a traditional sense; I define a parameter space navigated by the dancer while steering system affordances. Movements are multidimensional and variable; sound emerges from the interaction between human and system. Listening is mediated: the dancer perceives herself through system feedback, and I perceive her intentions through sensor readings, creating a dynamic, reciprocal relationship.

Uncertainty is intrinsic to the process. Noise, misreadings, latency, and occasional system failures are accepted as part of the work. Compositional decisions became anticipatory rather than prescriptive, with time structures shaped by the dancer's choices. The system triangulates agency between my intentions, her body, and computational interpretation, producing elastic, relational forms.

Mapping strategies were simplified over time. Large-scale movements often exceeded sensor thresholds, producing stuck or saturated values. I compensated by narrowing or dynamically adjusting ranges, using parallel mappings rather than multidimensional ones, and creating "safe zones" where gestures could not disrupt the system. Despite these adjustments, continuous sensor data—accelerometers, EMG signals, and other inputs—retained granular, fluctuating qualities that strongly influenced the sonic result.

Conceptually, the sensors redefined authorship. I could not fully plan or control sonic outcomes in advance. Form, time, and morphology were shaped by the dancer's real-time decisions rather than solely by compositional intent. The technological intermediary became a medium for negotiation, distributing responsibility for the work's outcome among dancer, system, and composer.

The socio-technical lineage of the tools also influenced the piece. Leap Motion originates from VR and gaming contexts; Myo stems from medical and prosthetic research. These origins subtly shaped the aesthetics, particularly in visual layers, creating environments of avatars, robots, and gamified elements. The sound world reflects this hybridity, balancing precision and instability.

Finally, the final sound of *Floating Pointers* emerges from a network of agents: myself, the dancer, the sensors, and computational readings. Visualisation of gestures and parameters was often insufficient, requiring supplementary visual aids to understand interaction patterns. The resulting sonic texture reflects the dancer's embodied decisions as much as the compositional framework, producing drifting, non-repetitive structures that evolve organically.

*Floating Pointers* demonstrates the interplay of control and emergence. It shows how shared instruments and HCI/HHI reshape authorship, reorganise time and form, and redefine compositional identity when movement—rather than notation or predetermined sound—drives the musical outcome.

## X.4. Departures

### FluCoMa analysis, corpus bias, and algorithmic gesture formation

*Departures* (for string orchestra) investigates how analytical tools—particularly the FluCoMa library integrated within SuperCollider—shape the emergence of articulation gestures and structural development. The use of FluCoMa was intentional: I was interested not only in its technical capacities but also in the biases inherent in its algorithms. In this sense, the software acted as a partner, generating constraints, unexpected relationships, and characteristic patterns that directly influenced the compositional outcome.

The piece is grounded in a corpus of conTimbre[29] recordings, which served both for spectral analysis and gestural extraction. FluCoMa's nearest-neighbour corpus exploration workflow encouraged an algorithmically informed approach to articulation sequences. The tool repeatedly highlighted acoustically plausible but sometimes aesthetically uneven relationships. I frequently worked around these automatic proximities, selecting less obvious neighbours for use in gesture chains. The resulting sequences emerged from negotiation between the algorithm's clustering behaviour and my own compositional judgement.

Harmonic verticals and transitional patterns were derived from perceptual categorisation of sound material. SuperCollider was used to generate quarter-tone harmonic structures, organised according to qualities such as "frozen," "weightless," "grounded", etc., rather than through conventional theoretical rules.

[29] https://www.contimbre.com/ , Accessed on January 7, 2026

Morphing, cross-fading and other processes were applied in SuperCollider to develop harmonic fields evolution.

The core workflow unfolded as follows: I curated the corpus, analysed the selected samples with FluCoMa, mapped them into different descriptor dimensions, and reduced dimensionality where appropriate. Nearest-neighbour searches generated sequences that were played back in SuperCollider while metadata—file names, articulation, and related parameters—was recorded in a text file. When a sequence finished, the last sample became the seed for the next iteration. I manually notated the resulting gestures from both the text file and audio recordings, analysed patterns that were atypical for my own compositional habits, and distributed them across the ensemble.

This process highlights the layered ecology of software in *Departures*. SuperCollider functioned as both instrument and environment, while FluCoMa served as analytical and generative partner. The nearest-neighbour analysis, descriptor-based clustering, and the chord builder created in SuperCollider all acted as co-creators, shaping articulation vocabulary, harmonic structures, and structural development.

The influence of the tools shaped compositional decisions in multiple ways. FluCoMa encouraged a top-down perspective: analysis determined structural possibilities before detailed sonic exploration. In contrast, the quarter-tone chord builder enabled bottom-up exploration, letting harmonic relationships emerge through iterative experimentation. Interface conditions also impacted temporal and gestural choices: the nearest-neighbour plotter displayed samples at fixed duration and amplitude, subtly biasing the resulting articulation sequences toward rhythmic regularity.

Tool biases were reflected in the material. Repetitions and clusters suggested by FluCoMa were not always musically desirable, requiring pruning or rearrangement. Limitations of the corpus—missing dynamic levels or certain playing techniques—introduced unevenness into the generated material, which became part of the work's character. Working with these constraints encouraged analytical listening, fostering attention to micro-variations in articulation and spectral detail, and informing decisions about sequence selection, reordering, adaptation and corpus curation.

Authorship in *Departures* is distributed between human and algorithmic agents. The software did not replace compositional judgment, but its tendencies shaped gesture, harmony, and form. The interaction required iterative negotiation: I balanced algorithmically suggested material with manual intervention to achieve perceptual and structural coherence.

Ultimately, *Departures* demonstrates how analytical and generative tools actively condition the compositional process. FluCoMa's descriptor-based clustering, nearest-neighbour logic, and inherent biases left tangible traces in the work. The piece is the product of an ongoing dialogue between my compositional intentions and the algorithmically mediated structures the tools suggested, showing how software can both challenge and extend creative agency.

## X.5. Cassandra v.1.0.3025

### LLM influence and algorithmic bias as dramaturgical material

*Cassandra* is a transmedia project shaped through my ongoing interaction with large language models, specifically ChatGPT[30]. The version *Cassandra v.1.0.3025* (interactive sound walk for singer, 3 musicians, portable electronics and 5 sound installations) examines how the LLM's behaviours, biases, structural limitations, and stylistic habits infiltrate compositional thinking and how these influences can be transformed into artistic material. Rather than concealing the AI's presence, the soundwalk exposes it, allowing the LLM's patterns to become visible and audible throughout the dramaturgy, sound processing, and textual surface.

The first realisation of *Cassandra v.1.0.3025* takes the form of an interactive soundwalk through an imagined "Museum of Artistic Predictions in 3025." This tour unfolds across the Austrian Sculpture Park, guided by Cassandra, a hybrid figure voiced and performed by a live singer whose vocal presence mingles with AI-derived phrasing and voice-cloned segments. As the audience follows Cassandra through the sequence of artistic "exhibits," each installation materialises a particular AI bias—biases that the LLM openly identified during the development phase.

[30] GPT-o3 and GPT 4.1, free version.

The structure of the walk reflects the imprint of AI-generated dramaturgical logic. My exchanges with the LLM frequently oscillated between coherent storytelling and sudden contextual breakdown, and this uneven rhythm influenced Cassandra’s character development dramaturgy and the soundwalk’s dramaturgy. This abruptness mirrors the AI’s tendency to lose its narrative thread and then default back into a neutral, instructive tone.

The LLM’s stylistic influence is particularly evident in the textual and vocal material. Early generated texts exhibited exaggerated emotional colouring, formulaic poetic gestures, and stylistic approximations based on the model’s interpretation of literary references provided during prompting. A persistent preference for ethereal, purified vocal qualities also appeared repeatedly in the model’s suggestions for sound processing. These tendencies were not adopted wholesale; instead, they were treated as pressures that had to be negotiated. The vocal writing deliberately oscillates between speech and song, voiced and unvoiced sound, linear utterance and mechanical repetition. Short looped recordings, fragmentation, stuck syllables, and reordered phonetic material are not only technical devices but compositional strategies that embody instability, interruption, and loss of agency.

The model also consistently proposed specific melodic and gestural tropes: narrow oscillations, smooth amplitude envelopes, pitch-stabilised contours, and reverb-heavy vocal aesthetics associated with contemporary synthetic voices. While these proposals informed the material, they were counterbalanced by transcribed contours derived from AI-cloned[31] vocal output and by composed speech patterns that resist smoothness. The resulting vocal language stages a tension between artificial cleanliness and human fragility, between controlled synthesis and audible breakdown.

Beyond vocal writing, the LLM influenced the design of performer–audience interaction. The model repeatedly generated generic interaction schemas oriented toward clarity, cooperation, and guidance. Rather than rejecting these templates, I used them as a structural starting point and then systematically destabilised them. Expected forms of interaction are withheld, misaligned, or rendered ambiguous. Performers may remain unresponsive where interaction is anticipated, guidance

[31] https://elevenlabs.io/docs/creative-platform/voices/voice-cloning . Accessed on January 7, 2026

may be suggested but not fulfilled, and leadership roles may shift or dissolve. These strategies expose the model's discomfort with ambiguity and its underlying assumption that agents should act harmoniously and transparently.

Sound processing throughout the work follows a similar logic of adoption and resistance. While the LLM often proposed conceptually rich descriptions of sound transformation, its concrete technical suggestions tended toward simplistic and conventional effects. Instead of implementing these suggestions directly, I treated them as conceptual triggers and expanded them into more complex acoustic and electroacoustic structures. Processes associated with memory, resonance, or identity are realised through layered feedback systems, spatially displaced delays, and conflicting signal streams rather than through literal effects. These strategies transform the model's reductive associations into unstable sonic situations that resist singular interpretation.

Authorship within *Cassandra v.1.0.3025* is continuously negotiated. The LLM's structural tendencies—such as symmetrical organisation, binary naming, and explanatory framing—shaped early versions of the dramaturgical architecture. However, the emotional and conceptual trajectory of the work diverges from the model's ideological preferences. Cassandra occupies states that the LLM struggles to sustain: prophetic uncertainty, political memory, trauma, silence, and glitch. Narrative coherence fractures at precisely those moments where the model would normally attempt to stabilise meaning. Neutrality replaces guidance, and clarity gives way to distance.

Working with ChatGPT involved both adoption and resistance. I adopted the model's fluency when structuring long sequences, its organisational clarity, and its neutral guide-like voice, which shaped Cassandra's instructional mode. But I resisted the system's aesthetic preferences by introducing glitch, fragmentation, acoustic instability, human breath, unresolvable ambiguity, and emotionally charged interruptions. In many parts of the walk, the neutrality of Cassandra's character turns into discomfort: Cassandra does not assist the audience, does not cooperate with performers, and does not explain how to interact beyond predetermined cues. The choreography of movement and sound stretches the AI's "helpfulness" until it becomes brittle.

Evaluating the artistic use of AI thus involves observing how the logics of sound transformation, the tension between acoustic and synthetic material, the shaping of narrative form, and the structural organisation all reveal traces of “tool thinking.” These traces are often most visible retrospectively, emerging from repetitions, preferred structures, oversights, and limitations in the system’s behaviour.

To sum up, *Cassandra v.1.0.3025* is not a showcase of AI-assisted composition but an exploration of how AI limits, simplifies, biases, and stimulates artistic thought. The soundwalk reveals these pressures and uses them consciously. The AI becomes part collaborator, part antagonist: a system whose patterns must be interpreted, reshaped, contradicted, and transformed. The resulting work makes the friction between human and machine intelligence not only visible and audible but also emotionally and conceptually central.

## X.6. What Is Left

### Algorithms of iterative inter-software transcription

*What is left* (for instrumental ensemble) is a piece built around the tension between human material and its software-interpreted reflections, using an iterative, modular workflow in which composed gestures are repeatedly transformed, transcribed, misread, and re-shaped by different algorithms. The work investigates how software interprets human data and how humans later reinterpret the software’s output. This circular process (or its parts)—composed → interpreted → re-composed → re-interpreted—became the core of the piece. The resulting score contains layers of both intentional design and encoded computational bias, creating a hybrid musical identity situated between human agency and machinic interpretation.

The computer music tools in this piece are not limited to synthesis or signal processing. Instead, the “instrument” consists of transcription algorithms, voice-splitting routines, quantisation processes, automatic harmonisers (as results of symbolic ring modulations), MIDI–MusicXML converters, and partial enhancers—each contributing structural changes to the music. The composition is structured in six main sections, built from combinations of composed (C, by human) and interpreted (I, generated by software) materials: CN–ICN, CM, CIN–IN–CM, Mel–

CCh–ICCh, CICh–ICh–ICM, and finally CM–CN–Mel. The first letter indicates whether the material originates from a composed or source; the second set of letters defines the musical object—motives (M), noise (N), melody (Mel), or chords (Ch). These labels represent the computational history of each segment, rather than adhering to conventional musical forms.

The workflow began with recordings of my own e-piano improvisation, stored as MIDI. I used several software environments to analyse this improvisation for patterns, biases, pitch-class tendencies, registral habits, and interval preferences. Max/MSP handled score-based operations such as Markov processing, automatic voice splitting, and various forms of auto-transcription and quantisation. SuperCollider functioned as an enhancer of partials for the stretched and then transcribed audio fragment and as a normaliser and "wrapper" for MIDI material using the SimpleMIDIFile Class. Reaper acted as the place where the audio was stretched and where mockups were assembled. Sibelius was used both to sketch audio exported from score drafts and to transcribe MIDI and MusicXML—often introducing errors. BasicPitch[32] provided an additional layer of transcription, and Python's[33] library was used to chordify the output, as well as analyse my own pitch-class and interval biases so that these biases could be amplified in subsequent human or software iterations.

The iterative loop was central: I allowed the software's interpretation to become a productive distortion. A composed segment was fed into an algorithm; its interpreted output, often quantised, reduced, or misread, was then shaped into new composed material, and this in turn became input for another round of software analysis. Every import–export cycle degraded the material in some way, and I deliberately used these degradations—lost notes, altered rhythms, narrowed registral spans—as musical material. Rather than correcting the software's mistakes immediately, I treated them as expressive deviations. For instance, Max's automatic split-voices object produced surprising groupings of the notes between 2 hands of a pianist or between different instruments/voices that I would not have invented myself, and I often preserved these as structural ideas. SuperCollider's partial enhancer generated chord-verticals that later appeared as harmonic pillars

[32] https://basicpitch.spotify.com/ . Accessed on January 7, 2026

[33] https://www.music21.org/music21docs/about/what.html . Accessed on January 7, 2026

in the score. Even the default timbral assumptions embedded in Sibelius's[34] MIDI transcription influenced rhythm and phrasing, since the quantisation tended toward regular patterns that I then had to "re-humanise."

The difference between composed and software-interpreted material became a structural engine. Interpreted segments often felt "decimated" by the software—flattened, compressed, or reshaped by its internal preferences—while composed responses restored nuance, gesture, and direction. Over many iterations, this produced a dialogue between human intentionality and algorithmic reduction. I treated software biases as collaborators: the algorithms mirrored my habits, but also exposed them by exaggeration. In some cases, material became clearer after interpretation; in others, it became crippled and had to be reinvented. The piece thus emerged from a continuous feedback loop in which I amplified, corrected, or resisted the biases embedded in the tools.

The software strongly shaped my compositional decision-making. I integrated algorithmic interpretation as a sequential step rather than an auxiliary tool, using it on both improvised and generated materials. Many musical decisions arose from the affordances of specific programs: Max's voice-splitting determined instrument assignments; Sibelius's MIDI interpretation influenced rhythmic notation; SuperCollider's enhancer created specific verticalities; and the cumulative import–export process progressively simplified textures. After every software pass, a significant cleanup phase was necessary. Errors—misalignments, quantisation artefacts, accidental transpositions—became part of the aesthetic. Instead of eliminating randomness, the workflow accepted and foregrounded it.

Conceptually, the process changed my understanding of authorship. I treated the tools as collaborators whose tendencies shaped form, harmonic space, and motivic evolution. This required rethinking the role of a score as a fixed object. Many sections were not "composed" in the traditional sense—they crystallised as the residue of algorithmic transformations. The iterative workflow also influenced my perception of time and morphology. A stretched audio became the basis for harmonic expansions; pitch-class tendencies revealed by *music21* reappeared in later iterations (also in my further improvisations); and the tension between

[34] https://www.avid.com/sibelius . Accessed on January 7, 2026

degraded, quantised outputs and their re-humanised reconstructions created a layered dramaturgy of loss, recovery, and transformation.

As an analytic practice, *What is left* exemplifies a self-reflective, annotated-portfolio approach. Each iteration documents where the tool interfered with or redirected the compositional path, allowing me to track how technology mediated authorship. The criteria that guided this reflection were the cyclical nature of the process (generation versus regeneration), the shifting composer–tool relationship (control versus resistance), the evolution of aesthetic language (from improvisatory fluidity to algorithmically filtered structure), and the distribution of agency between human intention and algorithmic behaviour.

*What is left* shows how an iterative, modular workflow can transform improvisation into a multi-layered instrumental composition in which software biases are not obstacles but essential compositional material. The piece is what remains after a long chain of interpretation, distortion, reinforcement, and re-composition—a musical residue shaped equally by human choices and computational processes.

## XI. How the tools rewrote my practice

What began as a belief in the neutrality and interchangeability of software shifted into the recognition that computer music tools do not simply execute ideas —they reorganise them. The six case studies made this influence unmistakable: SuperCollider's spectral logic and pattern thinking, FluCoMa's descriptor-based biases, Max's modular flow, the DAW's timeline perspective, notation software's quantisation defaults, and the interpretive distortions introduced at every import–export step. Understanding these influences not only makes me more aware, but it has fundamentally altered how I design the entire compositional process, watching how certain default behaviours reappeared across projects. In *PRISM,* the change of the creative coding environment and hardware controllers resulted in the new sound portrait of the piece and new gestural vocabulary; in *Projections* and *Timbral Approximations*, auto-transcribed spectral chords reorganised harmonic and textural thinking; in *Floating Pointers*, shared authorship emerged from working around software behaviour; in *Departures*, the corpus and analysis biases determined the shape of the material; in *Cassandra*, the bias became dramaturgical; in *What Is Left*, repeated interpretation loops created a combined identity of composer. Once I recognised these patterns, I began either actively avoiding the traps embedded in software design—the defaults that try to seduce me into writing the same gestures, structures, and textures, or using them explicitly as a creative means.

This reorientation became explicit in *Departures*, where working with FluCoMa inside SuperCollider revealed that analytical tools do not simply classify sound—they propose relationships of the biased in amplitude and duration corpus. Nearest-neighbour searches suggested articulatory continuities that were compositionally unusual, acoustically plausible but aesthetically unbalanced, and the resulting material emerged from negotiating with these tendencies rather than from pre-formed concepts. Analysis began to precede invention, and constraints became primary material rather than limitations to overcome.

A similar transformation appeared in *Floating Pointers*, where an iterative adaptation of the hardware mapping within negotiation between a composer/ computer music performer and a dancer resulted in the shared authorship.

In *Prism*, the software and hardware changes resulted in a changed sound, modes of interaction, and performative aspects of the piece.

In *Cassandra v.1.0.3025*, the software influence was linguistic rather than sonic. Interacting with a large language model introduced a behavioural bias: abrupt losses of narrative continuity, symmetrical conceptual framings, and a default instructional tone became audible and spatial. Instead of concealing these tendencies, the soundwalk exposed them. The dramaturgy mirrored the tool's logic through paired encounters, sudden resets of character, repeated thematic framings, and discomfort around ambiguity. Here, the software shaped dramaturgy, vocal and electronic identity, and audience interaction—both through sound processing suggestions and through narrative structure. The singer's voice negotiated between human fragility and the model's stylistic expectations, making the aesthetic emerge from resistance rather than adoption.

In *What Is Left*, I used software to reinterpret existing material across Max/ MSP, SuperCollider, Sibelius, Reaper, BasicPitch, and music21. Each import–export step introduced distortions—lost notes, simplified rhythms, unexpected voice-splitting, harmonic reinterpretations—which I began treating as compositional resources rather than errors. Over time, material accumulated a computational history inseparable from the piece itself, shifting authorship into alternating states of composition and interpretation.

Recognising these influences led to what I now call a modular software approach. Instead of letting a single environment dominate the creative logic of a piece, I assemble smaller software modules chosen for specific functions—corpus navigation, spectral interpretation, feedback analysis, rhythmic extraction, audio-to-MIDI degradation, score restructuring, mockup building. The underlying insight is that tools should not define my language; they should provide the conditions for articulating it.

This approach allowed me to bring my own compositional identity to the centre. I regained the ability to focus on harmonic precision, subtle gesture design, micro-timing, and dramaturgical placement without being absorbed by the thinking model of a single environment. The fluency I developed in SuperCollider made it possible to move between tools without losing expressive control; instead of feeling constrained, I now work within an ecology that reacts to my intentions.

Through this process, I learned that my artistic identity cannot depend on technological loyalty. The six case studies demonstrated that my compositional voice resides in my approach to time, harmony, gesture, and texture—not in Max/MSP, SuperCollider, or Pure Data. My micro-rhythmic textual fragmentation, feedback dramaturgy, chromatic-aberration metaphors, and modular shaping of articulation sequences translate across environments because they are ideas, not functions or patches.

Each computer music tool now functions as a collaborator rather than a system to adapt to. Translation between environments has become a creative act rather than a technical one: when material moves between Max/MSP, SuperCollider, Reaper, or notation, it inevitably transforms, revealing what is stable in my artistic voice and what is flexible.

To sustain this flexibility, I now invest in meta-skills—understanding spectra, debugging, mapping data, managing algorithmic complexity, and reading documentation to get a deeper insight into what each tool is meant to be for. These capacities allow the tool to become replaceable while the thinking remains intact. I also try to document my process rather than just the code: screenshots, mapping tables, signal-flow diagrams, and patch annotations preserve my reasoning rather than a software state.

Across these works, three transformations became clear: decision-making shifted from pre-planning to negotiation; form emerged from interaction rather than projection; and authorship became distributed. Working with these tools moved my position from controlling the system to responding to it—from using software to realise ideas, to using it to generate constraints, to allowing it to act as a structural and dramaturgical agent, and finally to accepting that a piece may become the residue of iterative translation rather than the execution of an initial plan. This reorientation did not eliminate intentionality but redistributed it: compositional thinking now includes analysing how the tool behaves and deciding when to adopt, suppress, or transform its tendencies.

## XII. Cognitive-dimensional analysis of my modular workflow

The Cognitive Dimensions framework (as adapted by C. Nash[35] for music-notation environments) is not a technical evaluation of software; it is a conceptual lens for examining how human thinking interacts with representational systems.[36] In composition, especially when working across multiple digital environments, the framework helps illuminate how visual, auditory, structural, and procedural features of a tool shape the way musical ideas emerge, evolve, and crystallise. By analysing attributes such as visibility, juxtaposability, viscosity, error-proneness, provisionality, and closeness of mapping, I can trace how each environment highlights some aspects of the material while obscuring others, and how these asymmetries guide my musical reasoning.

Applying this method to my own workflow reveals the subtle influences that are often overlooked in daily practice: why certain musical elements attract my attention; why others disappear in iterative transformations; why some errors become material while others are rejected; why specific musical structures become possible in one environment but not in another; and why some decisions become locked in prematurely due to the constraints of the tool. By shifting the

[35] Nash, „The Cognitive Dimensions of Music Notation“, p. 2

[36] Green and Petre, “Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework.”, p. 131–174

focus from “what I composed” to “how the tools made certain paths easier or harder,” this analysis exposes the cognitive ecology in which the music was created.

Most importantly, the Cognitive Dimensions approach allows me to examine the unconscious aspects of my process: the hidden defaults, habitual assumptions, representational biases, and workflow contingencies that silently shape the aesthetics of the final piece. Through this analysis, I gain a clearer understanding of the interplay between my compositional intentions and the behaviour of the tools I rely on, making explicit the tacit reasoning that underlies my modular computer-music practice.

I will try to evaluate my modular workflow on the example of the piece „*What is left*“, for which I have selectively applied the most suitable cognitive dimensions.

Because the piece „What is left“ relies on iterative reinterpretation—composed material becoming software-interpreted material and vice versa—the cognitive properties of each software layer have a tangible musical consequence.

• **Visibility** (how easy it is to view and find elements or parts of the music during editing) plays a crucial role in how I navigate this multi-step workflow. On the DAW timeline, visibility is middle to high: rendered items appear clearly, and I can track structure, density, and gesture placement at a glance. In contrast, visibility in SuperCollider and Max/MSP drops significantly; the text-based interface reveals logic but hides musical form, forcing me to switch constantly between code, listening, and mental imagination. Each step, therefore, foregrounds different aspects of the material. Reaper highlights macroscopic structures, while Max/MSP or SuperCollider reveal more microscopic regions of focus. This alternation between visibility and invisibility shapes my attention, often pushing me toward auditory verification when visual cues are insufficient. In Sibelius, it becomes extremely complicated again.

• **Juxtaposability** (how easy it is to compare elements within the music) is strongest on the DAW timeline. In Reaper, I can compare pre-rendered layers, sections, and timbral variants easily, which allows mockups to function reliably as pre-final structural evaluations before engraving the score. In earlier stages, however, juxtaposability is limited: Max/MSP patches, SuperCollider code, and intermediate MIDI files rarely coexist on one screen. As a result, form emerges through local sequential focus within a single tool rather than global comparison, and only once the material reaches the DAW do the layers become fully comparable.

• **Progressive evaluation**—how quickly I can test an idea—is fast at each individual stage, but difficult across stages of composing. Within Max/MSP or SuperCollider, immediate audition encourages small-scale improvisation and rapid trial-and-error, but travelling backwards from the mockup to an earlier tool breaks this flow. The feedback loop is strong locally yet fragile globally, which means improvisation happens inside each module, whereas planning is needed to transport material between them. The iterative ideas, distributed between several tools/environments, are rather complicated to test.

• **The balance between conciseness and diffuseness** (what is the balance between detail and overview) varies dramatically across tools. Reaper provides a macroscopic overview; Max/MSP encourages extremely microscopic manipulation; SuperCollider oscillates between the two. Sibelius can support both macroscopic for the temporal envelopes, and microscopic for single-event articulation or dynamics. This variation produces a cognitive stretch: I may get lost in details in one environment and later lose important details during transcription or export-import steps. The workflow forces me to shift repeatedly between high-level and low-level thinking, which in turn shapes the musical architecture—the large form is built in Reaper, while the detailed variations emerge in Max/MSP or SuperCollider.

• **Provisionality** (can I sketch and play with ideas without being precise about results) is a strength of the entire system. I feel safe experimenting in all environments, whether sketching in SuperCollider, improvising in Max/MSP, stretching material in Reaper, or auto-transcribing in BasicPitch. The workflow supports messy, temporary ideas; many fragile versions and partial states accumulate, allowing me to think through sound without committing prematurely. Again, when the ideas involve several environments to engage at once or in sequence, it could be rather complicated since the individual settings and sequences should be precisely tracked.

• **Secondary notation**—comments, sketches, annotations—is something I try to maintain across environments, although I often lose these notes as the project evolves.  Additionally, since 2023, I have analogue and digital diaries for each composition. Nevertheless, these informal traces make it possible to reconstruct previous steps if I decide to step backwards, even if that decision does not come easily. They also reveal my thinking over time, which supports reflective revisions.

• **Consistency** (are similar meanings represented in similar ways) across the environments is low, because each software uses different representations, metaphors, and logics. This inconsistency prevents me from relying on automatic habits and forces me to re-evaluate each element of the music at least twice. Far from being a problem, this repeated cognitive friction becomes a compositional method: inconsistency helps me rewire choices and articulate more deliberate decisions. That’s why a modular workflow inherently demands a significant cognitive load.

• **Viscosity** (is it easy to go back and make changes) becomes significant once the material is rendered in Reaper. After a mockup is consolidated, it is difficult to return to earlier stages without disrupting existing work. Revisions become costly and sometimes impossible without rebuilding the chain of transformations. This viscosity imposes a forward momentum that stabilises the composition at certain points.

• **Role expressiveness**—the ease of understanding what each part does—is mostly clear. Objects, code segments, and processing steps generally behave as their labels within environments imply. Ambiguity rarely frustrates me; instead, I use it when needed to provoke new ideas.

• **Premature commitment** (must edits be done in a fixed order, forcing planning ahead) is pronounced in moments of quantisation or any export–import across pieces of software. Grids, tempos, bar structures, and voice assignments often must be chosen earlier than I would prefer. These early decisions become difficult to revise later and shape the piece in ways I cannot fully foresee at the moment of commitment.

• **Error-proneness** (how easy it is to make annoying mistakes) is built directly into the aesthetics of the piece. Because the work relies on iterative interpretation, mistakes—quantisation artefacts, lost notes, misread intervals, mis-assigned voices—are not merely tolerated but actively used as compositional material. Some errors become expressive distortions incorporated into the structure; others must be repaired when they undermine coherence. The workflow treats accidents as both hazard and opportunity.

• Finally, **closeness of mapping** (does the environment match how I conceptualise the music) is relatively high: the environments correspond reasonably well to how I conceptualise sound and structure, though the mapping is rarely unified across tools. A more consistent symbolic representation within each step (as well as auto-saves of used presets) would help, but the gaps between tools often lead to innovation, pushing me to rethink my mental model when confronted with mismatches.

Together, these cognitive dimensions reveal that my modular workflow is not simply a chain of tools but a system of alternating constraints and affordances. Visibility and juxtaposability determine when I think in large form; conciseness and provisionality decide when I think in detail; viscosity determines when I commit; and error-proneness and inconsistency generate the hybrid material characteristic of my iterative practice. In this setup, the tools do not merely assist composition—they actively shape the music, the behaviour of the score, and my evolving aesthetic logic.

The modular workflow of this piece did not arise from conceptual planning but from necessity: the work depends on iterative reinterpretation, where rendered material becomes source material again, and this circular logic can only unfold across multiple environments. By examining the workflow through Cognitive Dimensions, I was able to understand why this structure was not merely practical but musically rich. Dimensions such as provisionality and error-proneness explain how the piece could incorporate fragile intermediate states and unintended artefacts as compositional material, while inconsistency and progressive evaluation account for the productive frictions that forced ideas to be manually re-examined rather than reproduced automatically. Visibility and juxtaposability clarified why large-scale form emerged only at certain stages, and viscosity revealed how moments of irreversible commitment stabilised the work. Through this reflection, I recognised that the modular workflow (which I use in my composition process) is not just suitable for this piece—it has become a preferred method in my wider practice because it distributes thinking across different representational logics, preventing premature closure and encouraging transformation rather than optimisation. The very dimensions that complicate the workflow—its inconsistency, provisionality, and controlled viscosity—now function as artistic strategies that enable more elastic, exploratory, and self-critical forms of composing.

## XII.1. Lifespan of the computer music tools.

### Software and hardware fragility

As highlighted in the case study of *PRISM*, the obsolescence of software can jeopardise an entire composition. This raises a fundamental question for composers: how should one invest time in learning tools that will remain relevant over the long term? A key strategy is to prioritise tools that cultivate transferable ways of thinking rather than merely offering specific features. Environments that foster conceptual understanding—such as signal-flow reasoning in Max/MSP, algorithmic thinking in SuperCollider, or principles of digital signal processing, data structures, and mapping strategies—teach paradigms that outlast particular software versions. While tools inevitably change, the underlying paradigms remain.

Equally important is the selection of tools within open ecosystems and communities. The relevance and longevity of a tool are rarely dictated by corporate agendas; rather, sustained artistic engagement ensures survival. Software like SuperCollider and Pure Data tend to endure because they are maintained by communities of artists rather than markets. Alongside choosing resilient tools, composers benefit from developing personal libraries of code, templates, and workflows. Accumulated scripts, Max/MSP abstractions, MIDI utilities, and analysis routines become an extension of the composer’s instrument, travelling across software updates and even into new environments.

A productive approach to learning involves a “core–periphery” strategy: deeply mastering a few core tools, such as a primary programming language, DAW, or notation environment, while maintaining light familiarity with emerging software, AI models, or plugins. This ensures both stability and adaptability. Understanding the history of the tools one uses is equally instructive. Tools with decades of community momentum—SuperCollider, Max/MSP, Csound, Pure Data

—are likely to persist in some form, even as their interfaces evolve. Investing in tools that support interoperability, through protocols such as OSC, MIDI, further future-proofs one's work by enabling communication across different systems.

A central concern is how to compose electronic works that remain performable ten or more years later, despite software evolution and system incompatibilities. Prioritising interoperable standards is crucial: OSC, MIDI, audio I/O protocols, JACK, VST/AU, and scripting bridges ensure that a piece does not depend on fragile, niche plugins with built-in expiration dates. Archiving everything meticulously—software versions, external units or UGens, operating system versions, screenshots of settings, simple "how-to-start" files, audio test files, and printed signal-flow diagrams—allows future performers and composers to reconstruct the work in a process akin to digital archaeology. Sure, it would be a luxury if the documentation process could be fully automated. Complementing this, creating a fallback version that renders critical audio processes offline, simplifies the patch to core actions, or provides a stereo playback alternative guarantees that the essential musical content survives alongside an ideal, interactive version.

Compositional structures benefit from modular, decoupled designs, where analysis, synthesis, and mapping are separable. This allows one element to be replaced without overhauling the entire system. Selecting frameworks that evolve slowly and transparently—such as SuperCollider, Pure Data, Csound, Faust, and Reaper scripts—reduces the risk of sudden incompatibility. Moreover, computer music works must be performer-readable rather than developer-readable; if musicians are required to act as system administrators to perform a piece, its longevity is severely compromised.

Software-specific dependency is inevitable in electronic composition, but it can be approached deliberately. Rather than avoiding dependency, composers can design it consciously, thinking in terms of replaceable building blocks. One can ask whether a particular filter, granulator, or feedback UGen is essential to the work, or whether any equivalent module could fulfil the same role. Separating the conceptual work from its specific implementation ensures that the musical intention—such as spectral morphing, density variation, or spatial trajectories—remains intact even when migrating to new tools. In this sense, software features should be treated as instruments with finite lifespans. Just as early tape techniques, analogue vocoders, and vintage samplers are reinterpreted with modern tools, the essence of a digital work can persist beyond the obsolescence of its original environment. Change thus becomes part of the ontology of electronic music: migration to new systems is not loss but continuation, akin to re-orchestration.

While software obsolescence is a significant challenge, hardware introduces a different layer of fragility, often with more severe implications for compositional longevity. Motion controllers such as Myo, Leap Motion, Kinect, and WiiMote embed very specific sensor grammars. The Myo armband structures gestures into defined poses, Leap Motion tracks palm and finger coordinates with millimetre precision, and Kinect maps skeletal positions with a characteristic smoothing behaviour. When these devices disappear, the embodied vocabulary they enabled vanishes alongside their APIs. A piece composed around the Myo's idiosyncratic gestures, for instance, loses its performative foundation when the hardware is no longer supported.

Commercial hardware further compounds fragility by enforcing proprietary pipelines, including firmware, company-maintained drivers, cloud-based calibration systems, discontinued SDKs, and operating-system-level permissions. A composition can "break" simply because an operating system drops a USB driver or a software development kit ceases to compile. Beyond technical fragility, hardware obsolescence affects the interaction paradigms themselves. Devices carry compositional assumptions: the Myo encouraged binary pose-based gestures, Leap Motion continuous mid-air sculpting, and Kinect full-body mapping. Once these tools disappear, the underlying assumptions become historical artefacts, and the artistic ideas entwined with them risk being lost. Unlike historical acoustic instruments, which can be repaired or reconstructed, digital instruments are often irretrievable.

This fragility directly impacts the identity of a work. Composers must decide whether a piece resides in its interaction logic or its sonic output, and whether gestures can be remapped without fundamentally altering the work. For motion-based compositions, the "instrument" is often the mapping between body and sensor; changing the sensor reshapes the choreography, score, and sonic logic.

Artists respond to hardware obsolescence through several strategies. One is migration: continuously re-implementing interaction models on successive hardware generations to preserve the original idea. Another is a hardware-agnostic design, where gestures are defined abstractly (e.g., “rotation” or “sudden extension”) rather than tied to device-specific features. A third strategy treats the work as a historical instrument, valuable precisely for its technological specificity. Finally, open-hardware approaches—Arduino- or ESP32-based controllers, OSC protocols, and open-source tracking solutions—reduce dependence on commercial ecosystems and extend the potential lifespan of a piece.

Hardware obsolescence, therefore, intersects with multiple dimensions of compositional identity, including temporal durability, embodied learning, material constraints, presets and SDK defaults, and techno-fluency gaps among performers. It is not merely a technical concern; it shapes choreographic authorship, performative embodiment, archival philosophy, and artistic identity. The ephemerality of commercial hardware forces composers to confront a fundamental aesthetic question: should works be designed to survive technological cycles, or is their transience an intrinsic part of their ontology?

## XIII. Conclusions

The concept of idiomatic writing, long discussed in instrumental composition, extends naturally to software environments. As A. McPherson and K. Tahıroğlu[37] note, “In every language, certain ideas and design patterns will be easier or more convenient than others… In the same way that idiomatic patterns of a traditional instrument affect improvisation and composition, what is idiomatic in a music language affects the in-situ exploration that characterises digital musical instrument design.” Each software system, therefore, carries its own idiomaticity: a set of strengths, preferred workflows, and cognitive affordances that shape how ideas are generated, developed, and realised. Creative flexibility depends on how well the composer can recognise and exploit these traits while navigating the inevitable biases of each environment.

[37] McPherson and Tahıroğlu, „Idiomatic Patterns and Aesthetic Influence in Computer Music Languages“ , p. 61

The analysis of my six compositions has revealed the multiple ways in which software environments influence both process and aesthetic outcome. Some tools shaped the material directly, through default sonic behaviours or structural tendencies; others guided the compositional mindset, reframing notions of control, gesture, or authorship. Across these works, I observed that ideas do not exist independently of the tools used to realise them: they evolve through a dialogue with the environment, transforming as the system's affordances, constraints, and idiomatic patterns assert themselves.

This insight extends to the evaluation of my modular workflow. By combining multiple software systems—text-based, visual, and hybrid environments—alongside hardware interfaces and algorithmic processes, I could distribute compositional tasks according to the strengths of each tool. This modularity allowed me to balance structured planning with exploratory, emergent behaviour. At the same time, managing multiple environments demanded conscious attention to cognitive load and technical complexity; it required making deliberate decisions about which constraints to embrace and which to resist. In practice, this meant that compositional choices were never purely aesthetic—they were inseparable from the characteristics of the tools themselves.

Key concepts emerge from this reflection: flexibility, idiomaticity, and meta-creation. Flexibility refers not only to the ability to switch between environments but also to the capacity to negotiate a productive dialogue with the system's inherent logic. Idiomaticity highlights the distinctive patterns and tendencies that each tool encourages, shaping both improvisation and design. Meta-creation describes the compositional stance in which the tool itself becomes an active participant, contributing constraints, suggestions, or surprises that inform creative decisions. Together, these concepts frame a perspective in which the software is not merely a vehicle for ideas but a collaborator, a mirror, and a co-author.

Recognising and analysing these influences has reshaped my approach to composing. Today, I design workflows that are modular, reflective, and adaptive: they balance intuitive exploration with conscious structural decision-making, accommodating the affordances and biases of each tool while maintaining a coherent artistic vision. In my future research, I will extend this inquiry, exploring how text-based, visual, modular, and AI-driven environments differently shape creative processes, outputs, and conceptual thinking. Understanding these relationships is both a technical and philosophical endeavour, revealing how tools mediate artistic agency, redefine compositional identity, and ultimately shape the music we imagine and create.

## XIV. References

### Articles and Papers

1. Bertelsen, Ole W., Morten Breinbjerg, and Søren Pold. “Emerging Materiality: Reflections on Creative Use of Software in Electronic Music Composition.” *Leonardo* 42, no. 3 (2009): 197–202.

2. Brown, Andrew R. “Tools and Outcomes: Computer Music Systems and Musical Directions.” In *Proceedings of the Australasian Computer Music Conference*, 9–16. Wellington, New Zealand, 1999.

3. Csikszentmihalyi, Mihaly. “Society, Culture, and Person: A Systems View of Creativity.” In *The Systems Model of Creativity*, 47–61. Dordrecht: Springer, 2014.

4. Dannenberg, Roger B. “Languages for Computer Music.” *Frontiers in Digital Humanities* 5 (2018): 26.

5. Green, Thomas R. G., and Marian Petre. “Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework.” *Journal of Visual Languages and Computing* 7 (1996): 131–174.

6. Green, Owen, Pierre Alexandre Tremblay, Ted Moore, James Bradbury, Jacob Hart, Alex Harker, and Gerard Roma. “Architecture about Dancing: Creating a Cross-Environment, Cross-Domain Framework for Creative Coding Musicians.” In *Proceedings of the Psychology of Programming Interest Group (PPIG) Conference*, 12–24. Milton Keynes: The Open University, 2022.

7. Magnusson, Thor. „Confessions of a live coder.“ In *Proceedings of the International Computer Music Conference*. 2011.

8. Marrington, Mark. “Paradigms of Music Software Interface Design and Musical Creativity.” In *Innovation in Music II*, edited by R. Hepworth-Sawyer, J. Hodgson, J. L. Paterson, and R. Toulson, 52–63. Shoreham-by-Sea: Future Technology Press, 2016.

9. McCartney, James. “Rethinking the Computer Music Language: SuperCollider.” *Computer Music Journal* 26, no. 4 (2002): 61–68.

10. McPherson, Andrew, and Koray Tahıroğlu. “Idiomatic Patterns and Aesthetic Influence in Computer Music Languages.” *Organised Sound* 25, no. 1 (2020): 53–63.

11. Nash, Chris. “The Cognitive Dimensions of Music Notation.” In *Proceedings of the International Conference on New Tools for Music Notation and Representation (TENOR)*. Paris, 2015.

12. Negrete-Yankelevich, Santiago and Nora Morales Zaragoza. “The apprentice framework: planning and assessing creativity.” *International Conference on Innovative Computing and Cloud Computing* (2014).

13. Tipei, Sever. “The Computer: A Composer’s Collaborator.” *Leonardo* 22, no. 2 (1989): 189–195. https://doi.org/10.2307/1575227.

14. Truax, Barry. “Musical Creativity and Complexity at the Threshold of the 21st Century.” *Interface* 21, no. 1 (1992): 29–42. https://doi.org/10.1080/09298219208570598.

## Books and Book Chapters

15. Brown, Andrew R. *Computers in Music Education: Amplifying Musicality*. New York: Routledge, 2007.

16. Farnell, Andy. *Designing Sound*. Cambridge, MA: MIT Press, 2010.

17. Magnusson, Thor. “Designing Constraints: Composing and Performing with Digital Musical Systems.” *Computer Music Journal* 34, no. 4 (2010): 62–73.

18. Nierhaus, Gerhard. “On Composers and Computers.” In *The Oxford Handbook of the Creative Process in Music*, edited by Nicolas Donin. Oxford: Oxford University Press, 2018.

19. Snape, Joe, and Georgina Born. “Max/MSP, Music Software, and the Mutual Mediation of Aesthetics and Digital Technologies.” In *Music and Digital Media: A Planetary Anthropology*, 220–266. London: UCL Press, 2022.

**Proceedings**

20. Brown, Andrew R. “Modes of Compositional Engagement.” In *Proceedings of the Australasian Computer Music Conference*. Brisbane, 2000.

21. De Campo, Alberto. “Loose Control, Gain Influence.” In *Proceedings of ICMC | SMC 2014*, September 14–20, 2014, Athens, Greece. https://quod.lib.umich.edu/i/icmc/bbp2372.2014.034/.

22. McCartney, James. “A Few Quick Notes on Opportunities and Pitfalls of the Application of Computers in Art and Music.” In *Proceedings of Ars Electronica*. Linz, 2003.

**Theses and Dissertations**

23. Kobzar, Alisa. *Sound Object*. Bachelor’s thesis, Universität für Musik und darstellende Kunst Graz, 2023. https://phaidra.kug.ac.at/o:128342/.

24. Pascale, Rodrigo Valente. *Aesthetic Implications of Using the Computer Medium in Music*. Master’s thesis, Western Michigan University, 2022. https://scholarworks.wmich.edu/masters_theses/5341.

**Other**

25. Kobzar, Alisa. *Timbral Approximation*. Project report. Universität für Musik und darstellende Kunst Graz, 2025. https://phaidra.kug.ac.at/o:137076.