AUF INS DIGITALE MATERIAL! EIN GESPRÄCH ZWISCHEN TECHNOLOGE UND DESIGNER

PATRICK TOBIAS FISCHER, CHRISTIAN ZÖLLNER

Patrick Tobias Fischer und Christian Zöllner sind Mitglieder des Medienkunst-Kollektivs VR/Urban. Sie entwickeln seit mehreren Jahren gemeinsam interaktive mediale Installationen für den öffentlichen Raum, die unter anderem im MoMA in New York ausgestellt wurden. Ihre Hintergründe sind jedoch sehr unterschiedlich: Zöllner ist Designer, Fischer Informatiker. Denk- und Herangehensweisen sind deshalb ab und an konträr.Kommuniziert und gearbeitet wird häufig anhand von Prototypen. Unterschiedliche Sichtweisen können so zu Potential werden und in diesem Spannungsfeld neue Ideen entstehen lassen. Die Interdisziplinarität bringt Dinge hervor, die in einer homogenen Gruppe nicht denk- und umsetzbar gewesen wären. Im Gespräch decken sie Gemeinsamkeiten und Unterschiede ihres Blickes auf Prototyping-Prozesse auf und diskutieren die verschiedenen Ansätze heutiger Prototyping-Toolkits.

Patrick Tobias Fischer hat einen Master in Informatik und promoviert derzeit im Fachgebiet Mensch–Computer Interaktion an der Universität Strathclyde, Glasgow, Schottland. Sein Forschungsschwerpunkt liegt auf der Schnittstelle zwischen sozialen und technischen Gesichtspunkten in ubiquitären und urbanen Umgebungen. Seine Arbeiten fokussieren vor allem auf die Kreation neuartiger Interfaces, die ihren Anwendungsbereich im öffentlichen Raum finden. 2011 arbeitete er zusammen mit Nicolas Villar bei Microsoft Research in Cambridge an der Hardwareplattform Gadgeteer.

Christian Zöllner studierte Produktgestaltung in Dresden, mit Arbeitsaufenthalten in Wien und Paris. Seit seiner Diplomarbeit (2007 in Kooperation mit dem Fraunhofer Institut IPK Berlin) beschäftigt er sich mit der Entwicklung von Werkzeugen zum räumlichen Skizzieren in virtuellen Umgebungen, wozu er international publizierte und ausstellte. Seit 2008 arbeitet Zöllner als künstlerischer Mitarbeiter an der Universität der Künste im Bereich »Entwerfen und Entwickeln im Design«.

Zöllner: Wie definierst du aus der Perspektive des Informatikers Modellbau?

Fischer: Für mich hängt Modellbau gedanklich mit U-Boot-Bausätzen zusammen, wie ich sie früher als Kind hatte. Das sind meistens Sets aus Kunststoffteilen, die nach Anleitung zusammengeklebt werden müssen, um sie im Anschluss auf das Regal zu stellen. Da ich früher auch RC-Car gefahren bin, kam ich neulich aber auf den Gedanken, dass Modellbau mehr ist als Prototyping. Im Bereich des Modellbaus gibt es zwei Phasen: Einmal die Phase des Bauens und zweitens die Phase der Nutzung. Man baut das RC-Car und anschließend fährt man es. Es gibt Leute, denen es mehr Freude bereitet, solche Modellautos zu fahren und Leute, die es bevorzugen, die Autos zu bauen. Und natürlich gibt es auch solche, die beides uneingeschränkt gern tun. Am Ende muss im Bereich des Modellbaus das Gebaute ziemlich perfekt funktionieren. Es ist kein Prototyp mehr, der versucht ein Funktionsprinzip zu erklären oder zu testen.

Zöllner: Aber ein Prototyp muss doch auch funktionieren?!

Fischer: Im Bereich der Informatik würde ich sagen: Ja, er muss funktionieren, weil anhand des Prototyps geprüft wird, ob ein Prinzip generell und in seinen Details funktioniert. Im Design ist das anders, da kann im Prototypen auch mal eine Ecke fehlen. Wenn beispielsweise im Entwurfsmodell einer Tasse ein Sprung ist, funktioniert der Prototyp trotzdem. Das ist, glaube ich, ein elementarer Unterschied.

Zöllner: So gibt es beispielsweise den Testing Prototype, in dem funktionale Eigenschaften geprüft werden und in Designkontexten gibt es Prototypen, die es erlauben, ästhetische und formale Qualitäten zu prüfen, ohne gleichzeitig die Funktionalität mit einzuschließen.

Fischer: Im Gegensatz zum Design werden in Software- sowie Hardwareentwicklungsprozessen meist erheblich kleinere Subabschnitte eines Systems geprüft. Man hat auch mehr mit einem symbolischen System zu tun und nicht mit einem physischen Objekt. Deswegen sind in erster Instanz mehr Untereinheiten zu prüfen und das Gesamtprodukt ist komplexer.

Zöllner: Du hast vorhin das Bauen von Prototypen von der Phase des Prüfens und Nutzens getrennt. Du stellst diese Abschnitte klar hintereinander: Gibt es in der Informatik auch Verschränkungen zwischen diesen Phasen?

Fischer: Entwicklungsprozesse sind eher mit einer Spirale vergleichbar. Mit klaren Phasen der Analyse, des Entwurfs, der Implementierung und des Tests und wieder zurück zur Analyse. Man baut also mehrere Prototypen bis zum Endprodukt. Und diese Prototypen sind immer andere Prototypen von integrierten Subsystemen.

Zöllner: Welche Qualitäten sollte ein Prototyp haben?

Fischer: Ganz oft ist es so, dass ein Prototyp der Kommunikation dient. Die Qualität ist also eine konzept- oder funktionserklärende. Er muss außerdem kundenorientiert kommunizieren, das heißt, er kann im Bereich der Software ein Papierprototyp sein. Man zeichnet klar auf, wie die GUI-Elemente aussehen und in der Software funktionieren. So lässt sich der aktuelle Auftragsstand dem Kunden oder den Mitarbeitern mit nicht technischem Hintergrund präsentieren und verifizieren.

Zöllner: Spielen bei deinen Prototypen formale Kriterien eine Rolle?

Fischer: Für mich spielt die formale Qualität eines Projektes eine relativ große Rolle. Für den Software-Entwickler generell sind formale Kriterien eher weniger interessant. Mir ist es wichtig, dass ich nicht an Projekten arbeite, von denen ich das Gefühl habe, dass sie schon einmal realisiert wurden. Ich versuche eher neue Sachen zu machen, dazu gehört auch das Erweitern von herkömmlichen, bekannten formalen Qualitäten.

Zöllner: Neu im Resultat oder neu im Prozess?

Fischer: Beides eigentlich. Wobei man bei Prozessen vorsichtig sein sollte. Es macht keinen Sinn, etablierte und funktionierende Prozesse und Methoden auszuklammern, nur weil sie nicht neu oder innovativ sind.

Zöllner: Mach doch bitte ein Beispiel.

Fischer: Es gibt zum Beispiel Framework- und Programmierregeln, die beschreiben, wie Programme aufgebaut werden müssen. Da würde man keinen neuen Weg gehen, weil das nur behindern würde.

Zöllner: Als Designer würde ich sagen, dass ich vielleicht aus Zeit- und Kenntnismangel Prozesse mal bewusst, mal unbewusst, anders angehe. Dabei entstehen dann Unschärfen im Modellbau, in denen die Chance für Innovation oder Neuerung liegt. Der Umgang mit Material, sei es digital oder physisch, birgt für mich immer Überraschungen. Ich neige dazu zu behaupten, dass Software dieses assoziative Arbeiten nicht zulässt, da die Regeln so stark sind.

Fischer: Es gibt für mich dazu drei Situationen. Erstens: Es ist klar, was gebaut werden soll. Das ist meist in kommerziellen, ausführenden Projekten im Bereich der Soft- und Hardwareentwicklung so. Dann gibt es den zweiten Weg: Es ist unklar, wie die Interaktion mit einer zur Verfügung stehenden Soft- und Hardware aussieht. Die Technologie ist aber so reizvoll, dass man unbedingt etwas mit ihr anstellen möchte. Das führt

dazu, dass man beginnt, mit der Technik herumzuspielen. Und dann ist es eigentlich so wie du sagst, dass plötzlich Fehler passieren, man einen falschen Datentyp gewählt hat und plötzlich etwas Abgefahrenes passiert. Mir ist es beim Experimentieren mit Displaymodulen passiert, dass durch einen Fehler in der Programmierung auf der Bildfläche dynamische Linienmuster entstanden sind, die ästhetisch sehr reizvoll waren. Diese Art Fehler können innovationsfördernd sein. Die dritte Vorgehensweise ist: Ich habe eine Vision und die soll in Hard- und Software konkretisiert werden.

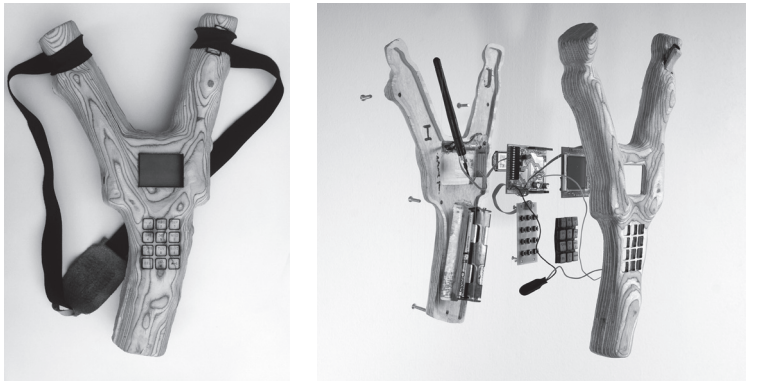
Zöllner: Gibt es nicht einen Unterschied zwischen einem Bug, also einem Fehler im Programm, der es nicht funktionieren lässt, und einem Gestaltungsspielraum? Mir selber ist es noch nie passiert, dass ich im Programmieren kreative Fehler gemacht habe. Vielleicht auch, weil es am Rechner die Möglichkeit des schwundfreien Zurücks gibt. Man kann immer zum Ausgangspunkt, Urpunkt, Protopunkt zurückgehen. Das erzeugt gegenüber dem händischen Modellbau in physischem Material ganz andere Iterationen im Arbeitsprozess, eine veränderte Prozessrealität.

Fischer: Das Reversible ist sicher ein wichtiger Aspekt. Wenn man etwas physisch baut oder entwirft, besteht natürlich die Möglichkeit, dass etwas durch eine Fehlentscheidung zerstört wird, oder weniger drastisch ausgedrückt: mehr Arbeit in der Korrektur erzeugt. Oder man muss tatsächlich von Neuem beginnen. Was auch nicht schlecht ist. Da lernt man das auch.

Zöllner: Kann man sich auch so ›ver-programmieren‹, dass es kein Zurück mehr gibt? Dass man nur mit einer kreativen Umweglösung wieder vorankommt?

Fischer: Wenn man eine gesamte Software-Architektur baut, kann man ein sogenanntes *refactoring* machen. Da spricht man aber schon von Projekten, die ›hausähnlich‹ sind, also in ihrer Komplexität einem Gebäude entsprechen, mit Stromleitungen, Wasserrohrsystemen, tragenden Wänden und freien Räumen.

Zöllner: Den Vergleich mit der Architektur finde ich interessant: Ein Architekt baut unterschiedliche Formen von Modellen. Einmal – auf dem Bauplatz - Verschalungsmodelle in 1:1, im Entwurfsbüro aber auch maßstäbliche Anschauungsmodelle der Gesamterscheiunung des Gebäudes. Gibt es vergleichbare Methoden im Programmieren? Fischer: In der Architektur ist der Architekt in persona getrennt vom Bauunternehmer. Der Graphik- oder Interaktionsdesigner ist auch getrennt vom



1

Die SMSlingshot ist eine technisch erweiterte Holzschleuder, die es erlaubt Textnachrichten einzugeben und über einen angesteuerten Projektor sichtbar an eine öffentliche Wand zu schießen. Sie besteht aus einer Holzhülle mit integriertem Mikrochip, Energieversorgung, Tastenfeld, Display und Funkmodem sowie Antenne.

Programmierer. In der Baustelle ist der Programmierer der Maurer, was heißen soll: Es sind getrennte Aufgabenfelder.

Zöllner: Nehmen wir unser gemeinsames SMSlingshot-Projekt als Beispiel: Die erste Platinenlösung war noch sehr vorläufig und wurde erst in weiteren Versionen spezialisiert und professionalisiert. Wie beschreibst du deine Arbeit an diesen Versionen und wo siehst du den Unterschied zu späteren Versionen der Schleuder, die als Platine und Programm bereits funktionierten, die ich aber auch noch als Prototypen bezeichnen würde, weil in ihnen noch soviel mehr Möglichkeitsraum steckt. (Abb. 1)

Fischer: Der Entwicklungsprozess der Platine ist ziemlich interessant, weil es die Herangehensweise einer größeren DIY-Bewegung abbildet. Es war doch so: Wir haben vorher eine Vision gehabt, wie der urbane Raum für eine freie Kommunikation zurückerobert werden kann. Das war kein kommerzielles oder zielorientiertes Projekt und keine Erkundung einer *awesome hardware*. Eher eine Idee, ohne wirkliche Ahnung wie man das realisieren soll. Es begann damit, dass wir Hardware benötigten. Dazu griffen wir auf die Prototypenplattform Arduino zurück. Bei Arduino handelt es sich um einen einfachen Chip, den man leicht bespielen kann. Von dem Arduino Board haben wir den Chip auf ein

Prototyping-Board gelötet und dort alles angeschlossen, was an externen Komponenten nötig war, plus eine Stromversorgung und einen Quarz. Das heißt, man hat das Board, das es schon gibt, dekomponiert, also auseinandergenommen und *reverse engineered* und ist zu einer Individuallösung gekommen. In den weiteren Schritten haben wir dann das Display hinzugefügt, das wir als fertiges Modul gekauft und von unnötigen Extrakomponenten befreit haben. Die Idee hinter dieser Art von Open Source Hardware ist, dass ich das komplexe Halbzeug auf das Wesentliche reduzieren und in mein System integrieren kann. Während des Entwicklungsprozesses musste ich auch immer wieder eine Risikoabschätzung machen, ob und wie das alles überhaupt funktionieren kann. Das führte im Weiteren zu einer Doppelplatinenlösung und noch weiter zu einem eigenen Modul, auf dem nur Chip und Display fix angebracht waren.

Zöllner: So entstand in einem frühen Stadium eine Lösung, die im Prinzip auch am Ende so gebaut wurde. Wie aber würde sich die weitere Entwicklung gestalten, wenn ein Großkunde auf uns zu käme, der die SMSlingshot marktreif produzieren möchte? Wir sind trotz aller Entwicklungsschritte noch immer im Prototypenstadium. Wie wäre es, wenn man den Prototyp als Vorserienmodell begreift?

Fischer: Wir wussten nicht, wie sich die Idee, Kurznachrichten visuell an eine Wand zu schießen, umsetzen lässt. Wir sind da sehr amateurhaft rangegangen. Während des Prozesses musste ich alles lernen, zum Beispiel wie Hardware zu konstruieren und zusammenzustecken ist. Wenn man das schon weiß, also Elektronik-Ingenieur ist, dann geht man anders vor, zum Beispiel sofort in Richtung einer Ein-Platinen-Lösung

Zöllner: Nach drei Iterationen sind auch wir dort gelandet …

Fischer: Ja. Ein Profi hätte aber nicht so lange experimentiert, sondern das Problem direkter gelöst, da er die Möglichkeiten von und den Umgang mit Hardware besser kennt. In solchen Fällen ist auch mehr Geld im Spiel. Professionelle Lösungen für Spezialanforderungen sind sehr teuer.

Zöllner: Die SMSlingshot ist ein Vorserienprodukt. Wären wir eine Firma, die professionell Embedded Systems herstellt, würden andere Prozesse greifen, weil andere Werkzeuge zum Einsatz kämen …

Fischer: Es ist natürlich so, dass Tools in der Software-Entwicklung sehr viel Geld kosten. Da kostet ein Tool gern mal 10.000 € und man befindet sich finanziell schon mal auf einer ganz anderen Ebene. Kauft man ein Tool, dann hat man meist nur einen Hammer, aber nichts zum Bohren. Man muss dann hinzukaufen. Will ich z.B. ein Blutdruckmessgerät bauen, dann bin ich eigentlich nur ein Integrator und kaufe Teile wie Pumpen etc. hinzu. In der Softwarebranche ist es derzeit so, dass sich solche Supply Chains herausbilden. Ich weiß genau, zu welcher Firma ich gehen muss, um dieses Softwaremodul zu bekommen. Es gibt zum Beispiel Firmen, die spezielle JAVA GUI Module liefern, wenn ich jetzt nicht die Standard Open Source Sachen verwenden will. Die integrieren das und ich verkaufe das Produkt.

Zöllner: Wie kommt es, dass professionelle Firmen so unglaublich viel Geld für Entwicklungstools ausgeben? Wieso tun die das, wenn die Open Source Tools inzwischen ähnlich leistungsfähig sind?

Fischer: Entwicklung kostet Geld. Und wenn ich spezielle Anwendungen brauche, dann muss ich die bezahlen. Die Alternative wäre, dass ich sie selbst entwickle. Da muss ich dann gefühlte 100 Mannjahre reinstekken.

Zöllner: Aber du redest doch jetzt von den Anwendungsmodulen. Ich meine die Entwicklungsumgebung. Was kann eine teure mehr als mein Arduino?

Fischer: Zum Beispiel Hardware Debugging, ich kann Breakpoints setzen, ich kann die Speicherpartitionierung nachvollziehen …

Zöllner: Aber das sind doch jetzt nur Analysefunktionen, die es dir erlauben, Qualitäten in deinem Prototypen expliziter auszulesen und zu testen. Hilft mir der Debugger auch, die Hardware besser auszureizen?

Fischer: Ja und das hat auch wieder Kostengründe. In der Serie hilft es mir nicht, wenn ich einen PC verwende, anstatt einfach einen Microchip zu nutzen. Der IC kostet mich nur Cents. Da wird dann auch meine Gewinnmarge höher.

Zöllner: Welche Methoden und damit verbundene Werkzeuge stehen für dich in den verschiedenen Phasen eines Entwicklungsprozesses prinzipiell zur Verfügung? Besonders interessiert mich die Art, wie du interaktive Benutzungskonzepte in Hardware umsetzt. Wie machst du das? Eher klar konstruktiv, also zielorientiert, oder offen und prozessorientiert?

Fischer: Wie ich es in den drei Vorgehensweisen bereits beschrieben habe, gehe ich klar konstruktiv vor, wenn ich einen kommerziellen Auftrag habe, wenn ich weiß, was ich zu tun habe und die Aufgabe also klar umrissen ist. Explorativ gehe ich vor, wenn ich eine neue Hardware nutze, deren Qualitäten ich interessant finde. Lösungsorientiert gehe ich vor, wenn ich eine Vision habe und die umsetzen möchte.

Zöllner: Welche Werkzeuge benutzt du bevorzugt? Das Repertoire, das hier und heute zur Verfügung steht, erscheint mir einerseits vielfältig und andererseits doch auch begrenzt. Was sind deine Favoriten?

Fischer: Mein Werkzeug ist zum Beispiel Eclipse. Wenn ich Daten analysieren möchte, dann mache ich das in Eclipse, weil ich mich mit der Programmierumgebung und der Programmiersprache Java gut auskenne. Java hat zudem den Vorteil, dass ich auf verschiedene Bibliotheken zurückgreifen kann, um eine Vielzahl von Aufgabenfeldern entsprechend zu bearbeiten.

Zöllner: Kannst du bei der Schilderung der Werkzeuge bitte stärker auf die Prozesse und Wege eingehen, die mit diesen Werkzeugen verbunden sind und die sie ermöglichen? Obwohl wir zusammen arbeiten, komme ich mir manchmal vor, als würdest du auf der anderen Seite der Welt arbeiten. Ich sehe immer nur Ergebnisse, nie aber den Weg dahin.

Fischer: Bei der Arbeit mit Hardware ziehe ich oft das Internet zu Rate und schaue, wie Andere ähnliche Probleme gelöst haben. Im Softwareentwicklungsprozess weiß ich eigentlich ziemlich genau was ich tue. Bei der Hardware kenne ich mich nicht so genau aus, was mich improvisieren lässt.

Zöllner: Das interessiert mich: Improvisation mit physischem, digitalem Material aus einem digitalen Hintergrund heraus. Wie machst du das?

Fischer: ›Heißkleber‹ ist ein gutes Werkzeug.

Zöllner: Ich empfinde den Umgang mit Heißleim als durchaus sympathisch, mit der Tendenz zum Pfusch. Generell würde ich sagen, ein Prototyp, dessen Komponenten mit Heißleim fixiert sind, ist ein *good enough prototype*, also ein Modell, das eine partielle Qualität darstellt und überprüfbar macht. Gibt es auch Heißkleber für Software?

Fischer: Ich glaube nicht. Man ignoriert eher das Problem in der Software und redet es sich schön.

Zöllner: In unserer Arbeit bemerke ich oft, dass du viel eher definierst, wohingegen ich mir viele Freiheitsgrade behalte. Liegt das an unseren unterschiedlichen Arbeitsmaterialien? Mein Holz lässt sich ad hoc schleifen, bohren, sägen. Dein IC lässt sich nur kompliziert programmieren und die Schaltkreise langwierig umstecken. Müsste man deine Werkzeuge verändern, damit Du ähnlich frei arbeiten kannst?

Fischer: Ich lasse mir ziemlich auch viele Freiheitsgrade. Ich lasse die Diskussion sehr lange zu. Man erhält sich dadurch die Möglichkeit, eine optimale Lösung zu finden. Aber ab einem bestimmten Punkt muss man Entscheidungen treffen. Sobald man anfängt zu bauen, wenn auch nur digital, sind die Freiheitsgrade weg. Man sollte dann nicht mehr im Code Designs ausprobieren, weil das zu viel Zeit kostet.

Zöllner: Wenn ich in unserer Zusammenarbeit im Prozess des Bauens umschwenke und beginne, Elemente meines Entwurfs zu verändern, zieht das auch Änderungen bspw. in den Interaktionsschemata nach sich. Wie kannst du darauf reagieren mit deiner Software? Musst du von vorn beginnen oder kannst du von vornherein so arbeiten, dass du in einem Prototypen bestimmte Möglichkeitsräume offen lassen kannst?

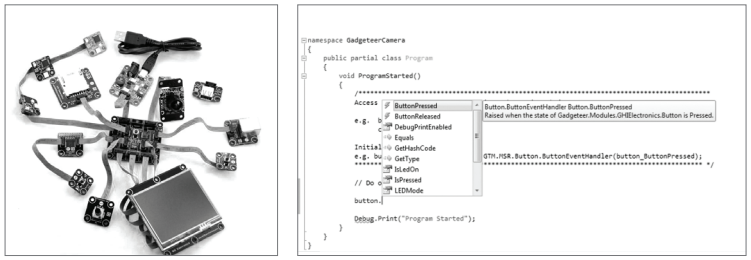
Fischer: Auch hier gibt es wieder mehrere Möglichkeiten. Es gibt einmal das Visuelle, also GUIs, dann gibt es Datenmodelle und Interaktionsabläufe. Das Schlimmste ist, wenn Interaktionsabläufe geändert werden. Das hat zur Folge, dass eventuell Datenmodelle geändert werden müssen. Das heißt meistens, dass von Grund auf das System geändert werden muss. Aber wenn man nur graphische Elemente ändern muss, dann geht das verhältnismäßig schnell.

Zöllner: Aber wenn ich nun das Interface nicht knopfbasiert, sondern reglerbasiert entwickeln möchte und anstatt zu leuchten soll das Objekt klingen? Kannst du dich in offenen Entwicklungsprozessen flexibel verhalten?

Fischer: Generell greife ich eher auf Bausätze zurück, die mit fertigen Modulen arbeiten. Ein Toolkit wäre gut, das es ermöglicht, diese formalen und technischen Komponenten wie Licht und Ton einfach auszutauschen. Das funktioniert nur durch Abstraktion, also wenn die Module so weit abstrahiert sind, dass man sie einfach zusammenstöpseln kann. Gadgeteer macht das beispielsweise schon. Ich kann ein Displaymodul einfach anstecken und über eine einfache Code-Zeile im Programm ansteuerbar machen. Oder ich klippe es wieder ab und hänge ein Soundmodul an, das mir die in den Code eingegebene Nachricht nicht visuell anzeigt, sondern vorliest. Das ist die Macht der Abstraktion. Was speziell dir helfen könnte, wenn du Probleme mit der Syntax von Programmiersprachen hast, das ist die automatische Codeergänzung. Microsoft macht das zum Beispiel ziemlich gut in seiner Programmierumgebung Visual Studio. Du weißt ungefähr, was für ein Objekt du erzeugen möchtest und beginnst erst einmal mit dem Buchstaben ›S‹ für Soundengine und es werden alle verfügbaren Objekte angezeigt die ebenfalls mit ›S‹ beginnen und unterstützen dich bei der Syntaxgenerierung. (Abb. 2)

Zöllner: Wenn Gadgeteer hilfreich sein kann bei modularen und assoziativen Vorgehensweisen im Entwurf, ist dann Arduino dafür weniger geeignet?

Fischer: Wenn man Amateur ist, ja. Generell spricht für den Ansatz von



2

Eine Auswahl an Modulen und Gadgeteer Hauptplatinen. Alle Platinen haben Löcher zur Befestigung bzw. zum zusammenstecken. Ein Flachbandkabel übertr. gt die Daten zwischen den Modulen. (rechts) Die komfortable Codeergänzung hilft beim ›Anprogrammieren‹der einzelnen Module.
(Quelle: http://www.netmf.com/gadgeteer/what-is-gadgeteer.aspx)

Gadgeteer die Schnelligkeit. Das hat Microsoft erkannt. In der *Sensors and Devices Group* bauen sie nun mal physische Prototypen und es ist natürlich viel einfacher, schnell ein paar fertige Komponenten zusammen zu stecken, als etwas in technischer Hinsicht schön, also effizient zu bauen. Die Zielgruppe von Gadgeteer ist also eher der Lehrbetrieb. Man lernt zwar weniger über den Aufbau der Komponenten, aber dafür einiges an C# und Interaktionsdesign und was diese Komponenten können in Bezug auf dynamische Systeme. Ich lerne nicht, wie und wo Ströme fließen und wie das Ganze physikalisch funktioniert. Das ist wegabstrahiert. Bei Arduino ist der Stromkreis noch erkl.rungswürdig, weil sonst das Projekt nicht funktionieren kann.

Zöllner: Begrenzt Gadgeteer damit den Entwerfenden nicht auch sehr stark?

Fischer: Bei Gadgeteer wird ein Modul mit bestimmten Funktionen geliefert. Man hat beispielsweise ein Kompassmodul und das gibt Norden, Süden, Osten und Westen aus. Das reicht nicht, wenn du Zwischengrade wie Nord-Nord-Ost haben möchtest. Das müsstest du hinzuprogrammieren. Wenn du das nicht kannst, dann bist du an dieser Stelle in diesem System gescheitert, egal wie modular es ist.

Zöllner: Könnte ich ein Projekt mit Gadgeteer beginnen und dann, wenn ich an Grenzen stoße, auf Arduino umsteigen?

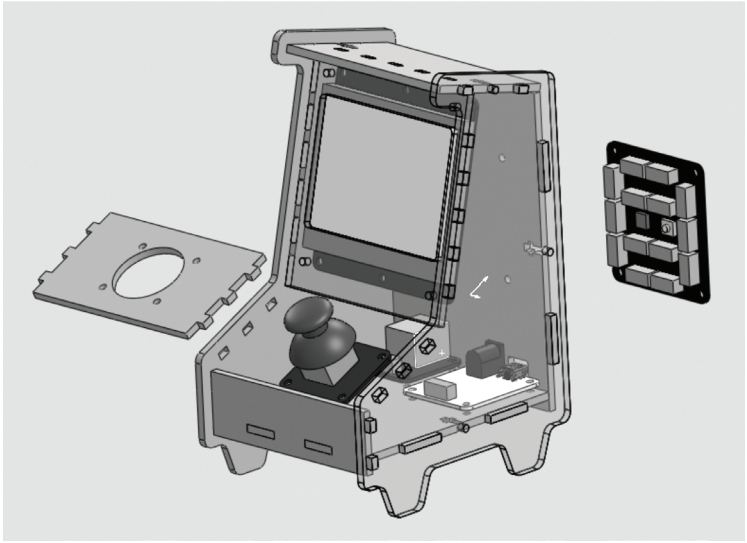
Fischer: Das Umsteigen auf Arduino macht da keinen Sinn. Auf ein anderes System umzusteigen ist noch viel umständlicher, als die Begrenzungen eines Systems zu überwinden oder die eigenen Fertigkeiten auszubauen. Du musst so oder so eine Abstraktionsebene hinuntersteigen im Code. Um bei dem Beispiel des Kompasssensors zu bleiben: Die Auflösung ist nur viergeteilt, also bei einer Wertspanne von 0-1023 ist bei 0 Nord, bei 255 Ost, bei 511 Süd und bei 767 West. Du kannst jetzt bei 895 ein Nord-Nord-West eintragen und wenn der Wert erreicht wird, dann zeigt es dir das an. Wenn dir die 1024 Untereinheiten nicht genug sind, dann musst du einen anderen fein aufgelösteren Sensor verwenden. Am Beispiel eines Displays würde man beispielsweise von »Schreibe den Text HALLO« hinuntersteigen auf »Mach mir eine Pixelanordnung in der Form eines H«. Und von dort lässt sich immer weiter in die Tiefe gehen. Es gibt aber auch physikalische Grenzen: Interrupts, das sind kurzfristige Programmunterbrechungen lassen sich z. B. nicht in beliebiger Anzahl setzen. Die müssen durch Hardware unterstützt sein. Weiterhin ist der Zugang zur Hardware bei manchen Systemen auch aus firmenpolitischen Gründen geblockt.

Zöllner: Weil es eben nicht wie Ardunio bis zur Basis Open Source ist, sondern ein von Microsoft eingeführtes Marktprodukt? Ist das die Diskussion »freie Elemente vs. modulares Konzept«? Ich finde Arduino immer noch interessant, auch wenn ich in meiner eigenen Arbeit sehr laienhaft damit umgehe und Arduino auch nur peripher in meine Lehre einbinde. Die Studierenden schätzen den einfachen Zugang zu Elektronik und Coding. Geht das auch mit Gadgeteer?

Fischer: Ja. Aber Gadgeteer hat andere Beschränkungen. Alle Aktuatoren und Sensoren sind auf PCB Platinen gelötet, um es schneller steckbar zu machen. Für den Designer äußert sich das in einem Objekt, das mindestens so groß ist wie eine Zigarettenschachtel und das als Volumen in den Entwurf eingearbeitet werden muss. Die Form wird also eventuell kompromittiert.

Zöllner: Gadgeteer bietet ein begrenztes Repertoire an Sensoren und Aktuatoren an, wohingegen sich Arduino an die ganze Welt ankoppeln lässt. Ich würde Arduino deshalb bevorzugen, weil es möglich ist, bereits existierende Technik anzusteuern, Stichwort Hardware Hacking. Arduino ist dann tatsächliche ein Interface, eine Vermittlungsstelle. Auch zwischen mir und der bereits fertig gebauten und verhausten Technikwelt. Über diesen Microcontroller kann man vieles aufbrechen.

Fischer: Das geht auch mit Gadgeteer. Für Gadgeteer gibt es Breakout-Boards, die es ermöglichen, mit Steckkontakten zu arbeiten, an die man seine losen Kabel steckt. Du hast mit Gadgeteer den Vorteil, dass



3

Bildausschnitt aus 123D. Das CAD Programm ist eine frei verfügbare Software zur Erstellung niederkomplexer 3D Geometrien. Es wurde von der Firma Autodesk 2009 entwickelt und basiert grundlegend auf der Entwicklungsumgebug Inventor, die in professionellen Konstruktionszusammenhängen eingesetzt wird. 123D richtet sich gezielt an Amateure.

du mit den Modulen datenreichere Projekte realisieren kannst. Du kannst leichter ein größeres Display ansteuern. Du hast mehr Pins zur Verfügung, mehr Speicher, größere Busbreite, eine höhere Taktzahl.

Zöllner: Könnte man also sagen, dass die erhöhte Leistungsfähigkeit bei Gadgeteer erkauft wurde durch eine geringere Offenheit und Integrationsmöglichkeit? (Abb. 3)

Fischer: Ich glaube, Gadgeteer bietet im Gegensatz zu Arduino eine Werkzeugkettenlösung an. Es lässt sich in Rapid Prototyping Prozesse integrieren. Beispielsweise mit 123D, das in einer Formdatenbank einzelne Module anbietet. Da lassen sich die Module virtuell zu einer formalen Lösung zusammenstecken. Dann gibt es zudem ein PCB Prototyping Tool und das Visual Studio habe ich bereits erwähnt. Und: Gadgeteer verfolgt eine Idee, mit der man relativ händisch an ein Design herangeht und eben physisch Prototypen bauen kann. Ähnlich dem Lego-Prinzip lassen sich Module gerüst.hnlich zu einem 3D Gebilde zusammenstecken.

Zöllner: Lass uns über die Gemeinsamkeiten und Unterschiede graphischer und textbasierter Programmierumgebungen reden. Ich habe manchmal Schwierigkeiten mit textbasierter Programmierung. Die Syntax erschließt sich mir nicht so schnell und Fehler sind schon durch vergessene Semikolons leicht erzeugt. Auch Arduino wird über Textzeilen programmiert und oft würde ich mir bildhaftere Ebenen wünschen. Gibt es Entwicklungen in diese Richtung?

Fischer: Ja. Beispielsweise gibt es eine automatische Codeergänzung von Eclipse und auch von Gadgeteer, die Fehler direkt anzeigt. Das macht die Entwicklungsumgebung von Arduino nicht. Arduino gibt dir auch keine Debugger Ausgabe, was es schwierig macht, Fehler im Programm zu finden.

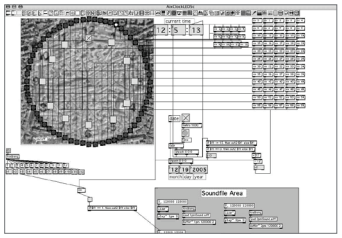
Zöllner: Code-Ergänzung und Darstellung von bestimmten Funktionalitäten in verschiedenen Farben sind sicherlich sehr hilfreich, aber von einem Bild, das mir die Übersicht über das Projekt ermöglicht, ist das noch weit entfernt.

Fischer: Aber das sind Ansätze. Ein Wortvorschlag ist auch eine Art Bild, ähnlich den chinesischen Schriftzeichen. MAX/MSP und das ähnliche VVVV gehen hier noch weiter und bauen auf graphischer Programmierung auf, so wie du sie ansprichst. Das sind erfolgreiche Lösungen, die besonders Medienkünstler einsetzen, da sie anscheinend intuitiver oder natürlicher zu verstehen sind. Die Abstraktion nimmt zu, geht ins Objekthafte und hilft, Fehler zu vermeiden. (Abb. 4)

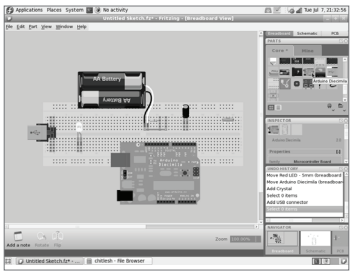
Zöllner: Warum wird dann überhaupt noch textbasiert programmiert?

Fischer: Textuell programmierst du, weil es übersichtlicher ist. In der Programmierung hast du meist eine Vielzahl an Objektklassen, die organisiert werden müssen. In MAX/MSP müssen das unterschiedliche Bilder sein. Das wird ab einem bestimmten Punkt schwer zu lesen, weil es keinen strukturierenden Codebaum mehr gibt. Ab einer bestimmten Projektgröße wird das Bildhafte nicht mehr handhabbar. Da muss man dann wieder auf Text umschwenken. Zum Ausprobieren eignet sich jedoch eher bildhaftes Herangehen. (Abb. 5)

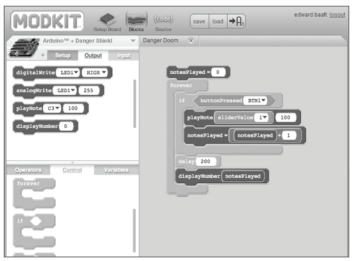
Zöllner: Fritzing stellt Schaltungen als Abbild der Steckung auf dem Breadboard dar und gibt sie als Schaltplan heraus. Gibt es so ein Prinzip auch für Software? Ein Programm, bei dem ich mir meine Funktionen bild- und objekthaft zusammenpuzzlen kann und am Ende wird mir das Ganze als Zeilencode ausgegeben? (Abb.6)



4 Jede Box verwirklicht eine Funktion in MAX/MSP. Am oberen Rand werden zu prozessierende Daten in einen Dateneingang hineingeleitet. Das Ergebnis wird dann am unteren Rand wieder ausgegeben. Verschiedene Funktionskästen können so mit virtuellen Drähtenkombiniert werden.



5 Bei Fritzing handelt es sich um eine Visualisierungssoftware, die es erlaubt, eine elektronische Schaltung als Steckdiagramm und als Schaltplan zu sehen. Auf diese Weise lassen sich elektronische Symbole und Schaltlogiken besser verstehen und Schaltungsentwürfe professionell kommunizieren. Fritzing wird an der FH Potsdam seit 2007 entwickelt und ist frei zugänglich.



6 Modkit ist ein Lehrprogramm für Grundschüler, mit dem diese die Hardwareplattform Arduino programmieren können. Die Idee wurde inspiriert durch Scratch, eine graphische Software der Lifelong Kindergarden Group am MIT Media Lab in Boston.

Fischer: Fritzing ist gut für Lehre und Dokumentation. Modkit ist ein gutes Beispiel, für das, was du beschreibst. If/Then-Schleifen werden in der Textprogrammierung als klammerartige Blöcke eingesetzt, was zu einem Erscheinungsbild führt, das an Tetris erinnert. Lego Mindstorm, die Programmierumgebung, um Legoroboter anzusteuern, hat ebenfalls einen Modus, in dem Funktionen zusammengestöpselt werden, der auch eher bildhaft aufgebaut ist. Bei Modkit sieht man den Code innerhalb dieser Puzzleteile, so lernt man dann auch gleichzeitig die Syntax. Vorteil ist, wie bei Fritzing, dass es verschiedene Ansichten (wie etwa Text und Bild) gibt.

Zöllner: Ich würde gern noch einmal zum Hardware Hacking kommen: Hardware Hacker nutzen bestehende Produkte und Objekte als Material, um daraus neue Objekte zu erzeugen; meist durch Umprogrammierung interner Schaltkreise und die Modifikation von Gehäusen. Wie müssen sich Produkte verändern, um diese Spielräume nicht nur zuzulassen, sondern zu fördern?

Fischer: Hier spielen Firmeninteressen eine Rolle. Firmen versuchen, ihre Arbeit zu schützen, das ist ganz klar. Das sieht man bei den Modchips für Spielekonsolen, die es erlauben, kopierte Videospiele zu nutzen. Der Einsatz dieser Chips, die ebenfalls Mikrocontroller wie Arduino sind, ist eine Änderung oder Befreiung der Hardware von Einschränkungen.

Zöllner: Aber es gibt Firmen wie beispielsweise Korg, die ihre Produkte schon offen für Hardwareveränderungen anlegen.

Fischer: Innovative Firmen gestalten Schnittstellen zu ihren Produkten offen und lassen Modifikationen zu. Zu Beginn der Computergeschiche war es ja so, dass Apple seine Schaltpläne offengelegt hat und auch jeder seinen eigenen PC bauen konnte. Ausgelieferte Apple Rechner konnten aufgeschraubt werden und ein angebrachtes Licht hat angezeigt, dass noch Strom auf der Maschine ist und man doch besser den Stecker ziehen sollte, bevor man anfängt Veränderungen vorzunehmen.

Zöllner: Schöne Anekdote! Es gibt auch das Korg Monotron, einen kleinen Mini- Synthesizer, der von Korg so angelegt ist, dass relevante Anknüpfungspunkte für externe Hardware als Lötpunkte sichtbar groß und handhabbar gemacht wurden. Auf dem Gerät ist das allerdings nicht explizit kommuniziert. Oder schau auf die Canon-Hacks. Glaubst du, dass uns eine Zeit bevorsteht, in denen die Grenzen der Hardware aufgemacht werden? (Abb. 7a+b)

Fischer: Eine Firma ist daran interessiert, dass das Produkt funktioniert,

weil es auch eine Produktgarantie geben muss. Und wenn dann der Fernseher nicht mehr geht, weil er durch die offene Schnittstelle umprogrammiert wurde, hat die Firma ein monetäres Problem. Eine Vielzahl an zurückgeschickten Fernsehern, bei denen der User die Firmware gecrasht hat, wäre der Fall.

Zöllner: Ist das dann eine Frage des Framework und der Gestaltung von Zugänglichkeit?

Fischer: Wie soll denn so etwas gehandhabt werden? Wenn ich einen Fernseher habe und der ist durch Inkompetenz kaputt gemacht worden, dann muss das das Verbrauchergesetz eines Staates, das mir ja 2 Jahre Garantie einräumt, abbilden.

Zöllner: Du sprichst oft von digitalem Material. Sind Arduino und Gadgeteer digitales Material?

Fischer: Ja. Seit den Radical Atoms Ideen von Hiroshi Ishii, bei denen Bits und Atome dicht beieinander liegen, ist die Trennung zwischen Objekt und digitaler Funktion kaum mehr auszumachen. Man kann von digitalem Material sprechen, egal ob es sich um Software oder Hardware handelt. Wichtig für die Schöpfer dieses Materials ist es, eine einfache Formbarkeit für den Designer herzustellen, so dass ihm ein greifbares und für ihn kreativ nutzbares Material zur Verfügung steht.

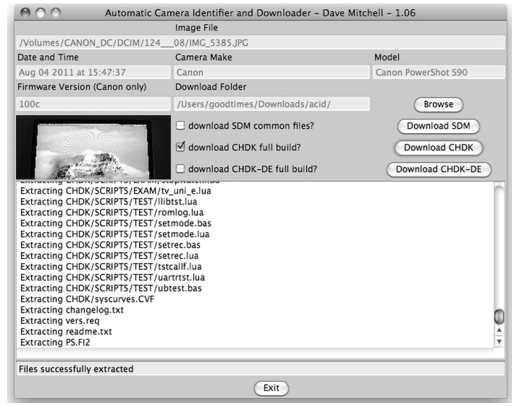
Zöllner: Kommunizierst du mit Ingenieuren anders als mit Designern? Einem Ingenieur könntest du mit UML Diagrammen sicher viel mehr sagen, als einem Designer, der eher in physisch handhabbaren Realitäten denkt …

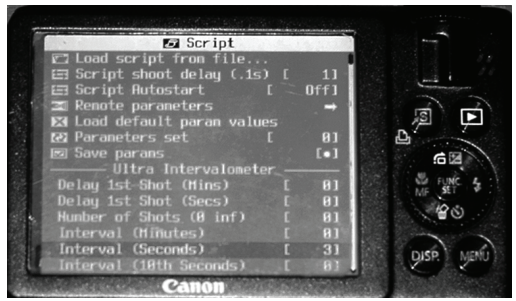
Fischer: Einem Designer gegenüber muss vieles nicht kommuniziert werden, weil es in meinen Aufgabenbereich fällt. Ich muss dir nicht sagen, wie ich meinen Model-View-Controller aufbaue, aber ich kann dir sagen, wo Probleme sind. Das hörst du ja sehr oft von mir: »Das geht (so) nicht (ganz)!« Typische Ingenieurantwort. Ich versuche so lange wie möglich nicht zu implementieren und mit Wizard-of-Oz, also Videoprototypen zu arbeiten, um bestimmte Teilprozesse zu visualisieren und erklärbar zu machen.

Zöllner: Du würdest in einem Informatiker-Team nicht anders vorgehen?

Fischer: In einem Informatikerteam würde ich technisch mehr pushen. Ich würde versuchen, technische Limits auszuloten. Das sind dann fachliche Details, die ich mit einem Designer nicht mehr aushandeln kann. Das sind andere Domänen, andere Expertisen.

Zöllner: Welche Expertisen teilen sich denn Informatiker und Designer?





7 CDHK ist eine Onlineplattform, auf der Programme gelistet werden, die es ermöglichen, reguläre CANON Digitalkameras einfach umzuprogrammieren. Über eine von CANON offen gelassene Softwareschnittstelle in der Kamera kann über die SD Karte das vorgegebene Programmrepertoire kreativ erweitert werden.

Beide Professionen widmen sich der Innovation, haben also auch ähnliche Lösungsstrategien, ihre Probleme auf kreative Weise lösen zu können. Ich rate jedem Designer, bei Gelegenheit mit einem Informatiker zusammen zu arbeiten. Technologie wird eine größere Rolle in unserer zukünftigen Gesellschaft spielen, und um diese human zu gestalten, müssen sich Designer als Experten der Menschlichkeit mit Informatikern an einen Tisch setzen und reden. Arduino hilft dem Designer, grundlegende elektronische und programmatische Probleme zu verstehen. Das führt gleichzeitig auch zu einer besseren und respektvolleren Kommunikation mit Informatikern. Und vice versa.