TRACING CONCEPTS

Version 1.0

In, Out and Through Computing

Edwin Gardner & Marcell Mars

TRACING CONCEPTS

In, Out and Through Computing

Edwin Gardner & Marcell Mars

While degrees, titles, institutes and schools constitute disciplinary boundaries and define professions, the ideas and concepts that circulate in a discipline are much more difficult, if not impossible, to cage. Ideas are borrowed and appropriated from one field to another. When adopted in new contexts concepts are mutated, bastardized and are re-utilized and, more often than not, depoliticized. When it comes to ideas we are all opportunists; we use them in our rhetoric and to shed new light on familiar problems. Emerging knowledge fields are especially hungry for ideas and prolific in generating new ones that travel across neighboring disciplines and beyond.

The arrival of the computer has produced a new knowledge field that was especially hungry for, and prolific in, concepts. Since computing is embedding itself ever deeper in the fabric of reality through a pervasive network of connected, communicating and sensing objects, it is worthwhile to trace and characterize the concepts going in, out and through computing. This is exactly what the *Tracing Concepts* project deals with.

The artificial environment of the computer is a place where particularly abstract concepts find fertile ground. Since "*software is perhaps the ultimate heterogeneous technology, it exists simultaneously as an idea, language, technology, and practice*" **NE2010**. In a very real way, a concept is literally put to work very differently than it would be in natural language, speech, thought or moving images. In a programming language, concepts become operative, mechanical cogs in the machinery of the Universal Machine.

How to use Tracing Concepts

This is the first iteration – version 1.0 – of Tracing Concepts. It focuses on the exchange of concepts between the fields of computing, architecture and philosophy, with a specific interest in the object-oriented paradigm that links these three fields together. The document is structured as a series of alphabetized encyclopedia entries, which can be navigated by the hyperlinks that connect all entries. We encourage you to follow the links, break out of the narrative, and start where you see fit rather than going from A to Z. Unlike an encyclopedia our entries are subjective, seeking to provoke and not insisting on a neutral position. Entries vary from definitions to extended quotations to full articles. Sources can be found at the end and are denoted by codes throughout. At tracingconcepts.net, the tracing continues.

Actor-Network-Theory (ANT) JL2011

Actor-Network-Theory is a disparate family of material-semiotic tools, sensibilities and methods of analysis that treat everything in the social and natural worlds as a continuously generated effect of the webs of relations within which they are located. It assumes that nothing has reality or form outside the enactment of those relations. Its studies explore and characterize the webs and the practices that carry them. Like other material-semiotic approaches, the actor-network approach thus describes the enactment of materially and discursively heterogeneous relations that produce and reshuffle all kinds of actors including objects, subjects, human beings, machines, animals, 'nature', ideas, organizations, inequalities, scales and sizes, and geographical arrangements.

Alexander, Christopher WP2011

Alexander (1936) is trained as an architect and mathematician. Reasoning that users know more about the buildings they need than any architect could, he produced and validated (in collaboration with Sarah Ishikawa and Murray Silverstein) a 'pattern language' designed to empower anyone to design and build at any scale (published as *A Pattern Language*) **CA1977**. Alexander has gradually positioned himself outside contemporary architectural discourse by his opposition to any contemporary architectural practice at large arguing instead for a return to pre-industrial building methods and for his fervent belief in an objective theory of beauty.
Alexander's *Notes on the Synthesis of Form* **CA1964** was required reading for researchers in computer science throughout the 1960s. It had an influence throughout the sixties and seventies on programming language design, modular programming, object-oriented programming, software engineering and other design methodologies. Alexander's mathematical concepts and orientation were similar to Edsger Dijkstra's influential *A Discipline of Programming.*

A Pattern Language's greatest influence in computer science is the design patterns movement. Alexander's philosophy of incremental, organic, coherent design also influenced the extreme programming movement. The 'wiki' was invented to allow work on programming design patterns (the wiki was named *WikiWikiWeb* and is the concept upon which *Wikipedia* is based). More recently, *The Nature of Order's* "deep geometrical structures" have been cited as having importance for object-oriented programming, particularly in *C++* (one of the most popular programming languages, for example Microsoft Windows is written in it).

Will Wright wrote that Alexander's work was influential in the origin of *The Sims* computer game, and in his later game, *Spore*.

Anthropocentric

see Correlationism

Anti-pattern WP2011 & WWW2011

In software engineering, an anti-pattern (or antipattern) is a pattern that may be commonly used but is ineffective and/or counterproductive in practice.

The term was coined in 1995 by Andrew Koenig, inspired by Gang of Four's book *Design Patterns*, which developed the concept of design patterns in the software field. The term was widely popularized three years later by the book *AntiPatterns*, which extended the use of the term beyond the field of software design and into general social interaction. An Anti-Pattern goes from a problem to a bad solution. A well formulated Anti-Pattern also tells you why the bad solution looks attractive (e.g., it actually works in some narrow context), why it turns out to be bad, and what positive patterns are applicable in its stead. According to Jim Coplien: *"an anti-pattern is something that looks like a good idea, but which backfires badly when applied."*

Often pejoratively named with clever oxymoronic neologisms, many anti-pattern ideas amount to little more than mistakes, rants, unsolvable problems, or bad practices to be avoided. Sometimes called pitfalls or dark patterns, this informal use of the term has come to refer to classes of commonly reinvented bad solutions to problems. Thus, many candidate anti-patterns under debate would not be formally considered anti-patterns. By formally describing repeated mistakes, one can recognize the forces that lead to their repetition and learn how others have gotten themselves out of these broken patterns.

Examples of anti-pattern are 'design by committee', 'groupthink', 'inner-platform effect', 'object orgy', 'golden hammer' etc.

Beirdo Mars

Masters of the Universe

There are only a very few fields whose practitioners imagine themselves to be creating the universe. They are able to imagine this because they have acquired the skill of working with a machine that holds the promise of performing this amazing ability. These practitioners are known as programmers, and the tool they wield is the Universal Machine – better known to us mere mortals as the computer.
Speaking about this Universal Machine in 1947, Alan Turing said, *"it can be shown that a single special machine of that type can be made to do the work of all. It could in fact be made to work as a model of any other machine."* This vision goes beyond mere mimicry among machines. Working with these machines programmers are able to see the world in this machine's image, all the while attempting to improve and augment the world through the language, analogies, images, and concepts that have been produced under the influence of the Universal Machine. Or when looking through the programmer's eyes and speaking in programmer speak: the world is a suboptimal place that can be optimized, and computers are the tools to do it with.

AMATEUR COMPUTER USERS GROUP NEWSLETTER HOMEBREW COMPUTER CLUB
Issue number two Fred Moore, editor, 558 Santa Cruz Ave., Menlo Park, Ca. 94025 April 12, 1975
Resource Sharing!
Altair
Intel
COMPUTER ART
T.V. TYPEWRITERS
FRED MOORE
CCD's
light pens

Cover of the Homebrew Computer Club Newsletter of 12 April 1972

Programming can improve the world incrementally by way of tiny steps or through utopian projects. Some of the holy grails of programming include:

- The dream of human-computer symbiosis and artificial intelligence.
- To give birth to a robot so intelligent that it will interact with humans as if it were one of them.
- Making a parallel, decentralized financial system, based on a cryptocurrency that cannot be controlled by central banks (thus proposing an alternative to fiat currency) and which can be traded anonymously with anyone.
- To make all the knowledge ever written accessible to everyone on the planet for free.
- To write a program that writes programs.
- To upload your mind into a machine, so it can do all of the above.

But you should be silent about your `search?q="holy grail of programming" - (About 16,600 results (0.19 seconds)`, because the world will think you are a beirdo .

The field of computer technology developed out of an interplay between science, engineering, business and the military, and has been characterized by a versatile group of experts and amateurs. Unfortunately, not only popular culture but also historiography tries to reduce the history of the computer boys to a small set of brilliant inventors. The computer boys remain an elusive group to trace, as are the many contributions of hackers, end users, operators, system administrators, geeks, nerds, wizards, gurus and other 'invisible technicians' who are often left unaccounted. Without them we would have but a very small subset of today's computer infrastructure.

From the early nineteen-seventies, once computer processors (which filled up entire rooms) evolved into their 'micro' relatives (which you could hold between your fingers), the world was introduced to an army of passionate amateurs, working in their garages, living rooms or even sitting on their beds, computer on their lap, hacking things done. To play for sake of play. To dream and share their love for computers with their fellow computer hobbyists – who usually wore beards too.

While technology and science have always been seen as a means to an end in cultural production, it was becoming apparent from the mid-twentieth century onwards that sound and video synthesis, recording and broadcasting technology, and computers and digital networks were themselves sources of cultural production, not just mediators of information. Computers have produced a very vibrant culture within the fields of science and technology, ranging from bureaucratic boredom to fantastic eccentricity. At first the cultural manifestations remained obscure and unpopular, safely confined to the domain of nerds, but the nerds have steadily made their way to mainstream popular culture.

For the past twenty years the computer boys' play of constructing a reality in between the two cultures of science and the humanities has made them the first-class citizens of contemporary artistic and cultural production: forensics in TV shows, conspiracy theories in documentaries, comics, science fiction and fantasy in Hollywood blockbusters, or entire new industries like video gaming – whose revenue rivals that of the movie and music industry and radically outpaces it in growth. As Kevin Kelly puts it, a *"pop culture based in technology, for technology. Call it nerd culture"* simply became cool. This 'third culture', as Kelly and others like John Brockman and Nigel Cross have dubbed it, represents identifiably different aims, values and methods of knowledge production than the other two cultural fields, science and the humanities.

Yet a regular problem is that a mastery of technology – the true essence of the third culture – is not being translated to a very wide audience. Technology continues to be perceived as a kind of black magic, something that happens behind the hectic animations of the bespectacled geek's screen.

Up until personal computers became connected to a global digital network, technological change took place through the emergence of relatively specific technologies (electricity, radio, television, transistors, video and tape recorder, and communications satellites, to name a few). Once the world of technology fuses with the digital network, soon each device will communicate with every other device in the network of (all) networks and the tool for making tools will slowly transform into software: the development of software will become metonymical for all technological development. Understanding what software is as well as the context in which it is developed is as much a privilege as it was to be literate in the sixteenth century. As Marshall McLuhan said, *"If we understand the revolutionary transformations caused by new media, we can anticipate and control them; but if we continue in our self-induced subliminal trance, we will be their slaves."* **MM1969** This is a warning that echoes in Douglas Rushkoff's *Program or be Programmed.* **DR2010**

Software is a socio-technical system in which computing technology meets social relationships, organizational politics, and personal agendas. Every time an organization starts to implement software it will need to restructure itself in order to accommodate new procedures, flows of information, social relations, corporate memory, monitoring, control, and demand to understand the new system as a whole. That process binds together, as Nathan Ensmenger writes, *"machines, people, and processes in an inextricably interconnected and interdependent system"* which never goes without *"conflict, negotiation, disputes over professional authority, and the conflation of social, political, and technological agendas. Software is perhaps the ultimate heterogeneous technology. It exists simultaneously as an idea, language, technology, and practice."* **NE2010**

Building Code

see Programming Language

Building Information Modeling (BIM)

Gardner

One Building Said To The Other

The involvement of the computer in the building industry has catered so far to every expertise separately. Every specialist works with their own software package, their own methods for calculation, analysis or design, and each with their own proprietary file formats. Up until recently there were no frameworks or models in which all these specialists could integrate their work. Information flows and formats are predominantly closed due to proprietary software, copyright issues, and disclaimers, and a general reluctance and distrust prevents participants in the building process from easily sharing information. In 2004 it was estimated that the U.S. building industry lost close to $16 billion per year due to inadequate interoperability highlighting *"the highly fragmented nature of the industry, the industry's continued paper-based business practices, a lack of standardization, and inconsistent technology adoption among stakeholders"*. **WP2011**

Building Information Modeling (BIM) is the post-Computer Aided Design (CAD) paradigm for how the computer can be employed in the building industry. It is an approach that could overcome the industry's information asymmetry problems. Charles M. Eastman at Georgia Tech coined the term BIM; his theory is based on a view that the term BIM 'Building Information Model' is basically the same as 'Building Product Model' which Eastman used extensively in his book **CE1999** and papers since the late 1970s. ('Product model' means 'data model' or 'information model' in engineering.) **WP2011** However, BIM is currently developing off the radar from the architectural avant-garde and academia who are more focused on digital fabrication and algorithmic-driven form-finding. BIM is an innovation that occupies the more corporate sectors of the building industry – the business of big projects and full-service offices. It is currently mostly used as a way of having one singular model from which all kinds of drawings for a project can be extracted, and as a way to manage and (re)use building components and systems across projects. Though, the BIM project has much more in store, it constitutes a radically different way of dealing with the built environment. With BIM programming is entering the construction industry at an industrial scale. This opens up a potential far greater than the algorithmic form-finding experiments of the architectural avant-garde. This is a potential we should seize and bring into the heart of our schools and discourse, one that should not be left to the corporate world alone. BIM's potential is that it constitutes a building as an entity in a vastly interconnected network in which it lives and engages with other entities in real time. The building becomes an object in an object-oriented world.

BIM defines the building as a node of information, a networked object, potentially a technologically enabled Gordian knot so to speak. The building-object is born with the act of making a file: it is simply given a name. Thus this object starts its life as an empty vessel at first vague and undefined and then begins to accrete information and definition over time, across various fields of knowledge. Bundles of data – the objects that are within the BIM-object or communicate with it – can be product-specific, void-space oriented, climatic data, GIS data, real estate taxes, cost calculations, energy data. Eventually, what began as a file turns into a virtual building with all its parts, nuts and bolts defined: a model that allows for simulating the flow of people, energy and matter. Using simulation scenarios can be quickly tested including the shearing of layers, maintenance costs, and perhaps even how material ages and acquire patina.

The virtual building then gradually becomes actualized on the construction site and the model is updated in real-time with the construction process. Changes are incorporated when necessary and market information on product and labor costs is verified and updated.

The life of the BIM model does not end when the building is handed over to its owner, however. Supported by the smart-sensor revolution, the virtual building will continue a parallel life to the actual building, becoming its information-laden avatar, accumulating data about usage, energy and the building's overall health. It will be able to indicate when maintenance is required, where problems are located, what specific replacement parts will be needed. The building and its avatar will make the real estate market more transparent and the problem of hidden deficiencies will be minimized. Imagine a real estate market in which you have a BIM model to go with the house; it might even become a requirement to trade real estate. Gradually the entire existing building stock would become virtualized.

During its lifespan the building will carry its growing history with it and, analogous to the accumulated data, it would will be able to make predictions about the future. It will have expectations. Buildings will be networked into a web and grouped into families, species, streets, neighborhoods, quarters, cities, regions, and countries. Each will become an object in its own right into which data from lower-scale objects will be aggregated;, each object will learn not only from its peers, but also from objects up and down the hierarchy – from leagues of nations to the microscopic droplets of moisture in the wall. These are the promises of BIM, an object-oriented building code, a programming language for the built environment, a Building Code.

BIMstorm

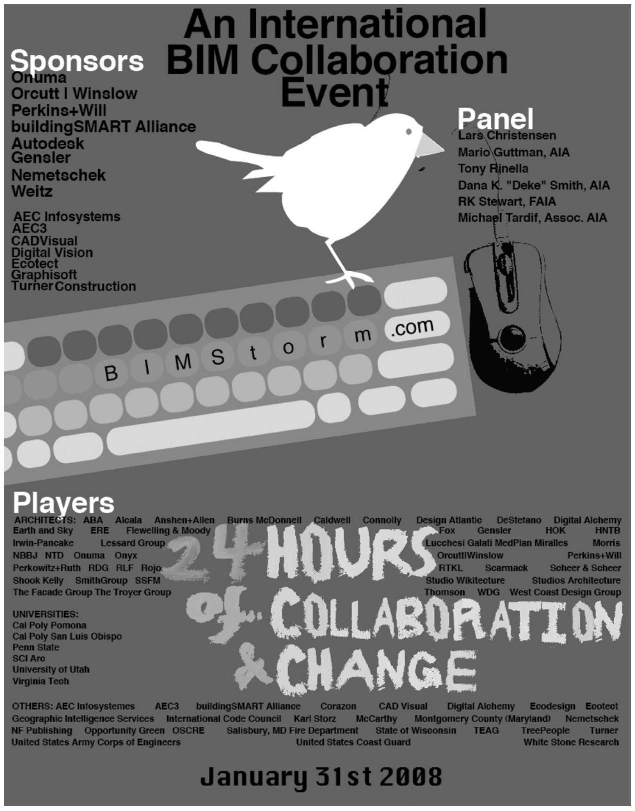
Gardner

Massively Multiplayer Online Building-Fest

From Silo-Culture to Share-Culture

BIMStorm is like a twenty-four-hour, worldwide LAN party for architects, engineers, technologists, consultants, and various experts from the building industry working on a particular site . BIMStorm has been held in cities across the globe including Los Angeles, New Orleans, London, Tokyo and Rotterdam. BIMStorm is structured according to a kind of unconference model, promoting self-organization, knowledge and skill sharing. This (an ethos was underscored by its the Woodstock-inspired poster for BIMStorm LAX in 2008). In an industry conditioned to protect specialist information from falling into the hands of others in the building industry's information silo, this gathering represents a culture alien to the industry. Everything is shared completely and open-sourced to peers and other experts. Here is a description by the initiator of BIMStorm Kimon Onuma: *"During the early morning hours of January 31, 2008, 'BIMmers' from the east to the west began collaborating. By noon, engineering teams from Honolulu to Manila engaged their efforts to provide structural support on a 54-story building. Multiple buildings and fire stations were located in the BIMStorm arena. While many U.S. teams closed for the night, teams in Hawaii, Asia and Europe picked up the project and designed the HVAC and structural systems. For the first time, global 'BIMmers' reacted much like stock market investors. These requests were then picked up by teams to resolve the design and placed on sites. Data was opened in energy analysis tools to generate calculations and graphics. Connecting the dots from early design through to the 20-year life-cycle was possible by sharing design decisions with many different experts and software. Building code checking using International Code Council rules, happened in parallel."* JBIM2008

Onuma is the man behind OPS (Onuma Planning System), a system which coordinates and shares all the BIM data over the Cloud. These shared efforts were brought together visually in GoogleEarth where the apartment blocks and skyscrapers grew out of a city hotbed over the course of twenty-four hours. It was like *SimCity*, but instead of adjusting the variables of an intricate algorithm, it was being generated by a human collective. OPS provided the platform, but what really made the exchange possible was the adoption of open standards such that one individual's tools could talk to everyone else's – a digital lingua franca. Cycles of feedback, design and consequence, analysis and adjustment could now feed each other across participating knowledge fields. A criticism begins to emerge: is this not a threat to the craft of design? When design processes are hybridized by programming, the character of design will certainly will change. But the opportunity is that since architects and engineers are the highly educated knowledge-wranglers of the industry – the industry's digital-natives, so to speak – they have a head start. A head start at reclaiming lost terrain in the building process, over which one can hear the architect so often lament. The precondition, however, is that architects not retreat into the mystique of craft, but make an effort to systematize and organize their knowledge so that it becomes easily utilizable. Socio-spatial Gordian knots embedded in neighborhoods and cities will have to become socio-spatial Gordian *nodes* in a *network*. Craft has always been about mastering technology. Craft will always be there; technology will always change.

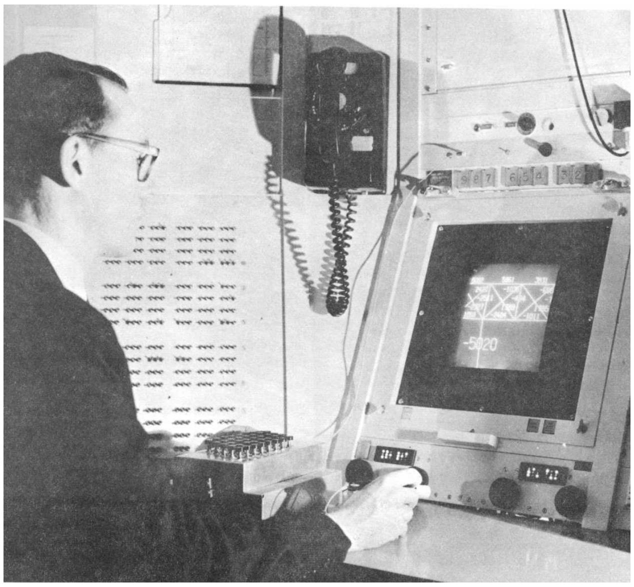

An International BIM Collaboration Event
Sponsors
Onuma
Orcutt I Winslow
Perkins+Will
buildingSMART Alliance
Autodesk
Gensler
Nemetschek
Weitz
AEC Infosystems
AEC3
CADVisual
Digital Vision
Ecotect
Graphisoft
Turner Construction
Panel
Lars Christensen
Mario Guttman, AIA
Tony Rinella
Dana K. "Deke" Smith, AIA
RK Stewart, FAIA
Michael Tardif, Assoc. AIA
BIMStorm .com
Players
ARCHITECTS: ABA Alcala Anshen+Allen Burns McDonnell Caldwell Connolly Design Atlantic DeStefano Digital Alchemy Earth and Sky ERE Flewelling & Moody Fox Gensler HOK HNTB Irwin-Pancake Lessard Group Lucchesi Galati MedPlan Miralles Morris NBBJ NTD Onuma Onyx OrcuttIWinslow Perkins+Will Perkowitz+Ruth RDG RLF Rojo RTKL Scarmack Scheer & Scheer Shook Kelly SmithGroup SSFM Studio Wikitecture Studios Architecture The Facade Group The Troyer Group Thomson WDG West Coast Design Group
UNIVERSITIES:
Cal Poly Pomona
Cal Poly San Luis Obispo
Penn State
SCI Arc
University of Utah
Virginia Tech
24 HOURS of COLLABORATION & CHANGE
OTHERS: AEC Infosystemes AEC3 buildingSMART Alliance Corazon CAD Visual Digital Alchemy Ecodesign Ecotect Geographic Intelligence Services International Code Council Karl Storz McCarthy Montgomery County (Maryland) Nemetschek NF Publishing Opportunity Green OSCRE Salisbury, MD Fire Department State of Wisconsin TEAG TreePeople Turner United States Army Corps of Engineers United States Coast Guard White Stone Research
January 31st 2008

Poster for BIMStorm LAX 2008

CAD

WP2011

Computer Aided Drafting, or Design (CAD) began with Sketchpad SketchPad (aka Robot Draftsman) was a revolutionary computer program written by Ivan Sutherland in 1963 in the course of his PhD thesis, for which he received the Turing Award in 1988. It helped change the way people interact with computers. Sketchpad is considered the ancestor of modern computer-aided drafting programs as well as a major breakthrough in the development of computer graphics in general. For example, Graphic User Interface (GUI) was derived from Sketchpad as well as modern object-oriented programming. Ivan Sutherland demonstrated that computer graphics could be used for both artistic and technical purposes, in addition to showing a novel method of human computer interaction..



SketchPad in use. On the display part of a bridge. Ivan Sutherland is holding the Light pen. The push buttons used to control specific drawing functions are on the box in front of the author. Part of the bank of toggle switches can be seen behind the author. The size and position of the part of the total picture seen on the display is obtained through the four black knobs just above the tabl

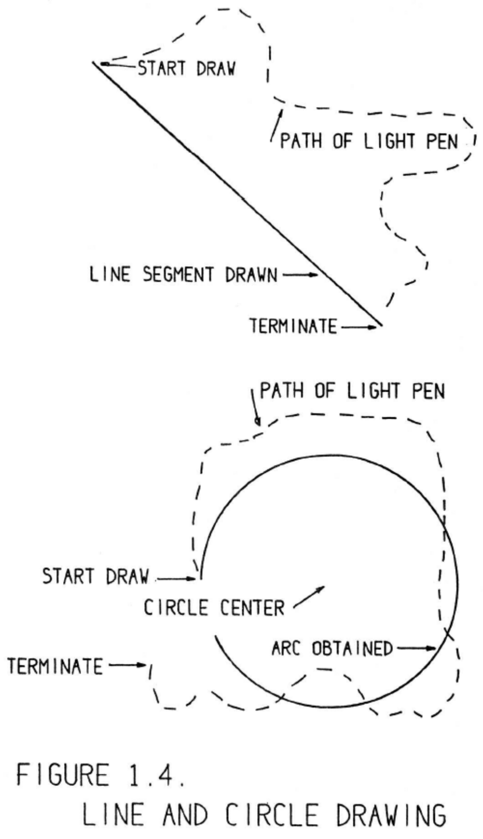
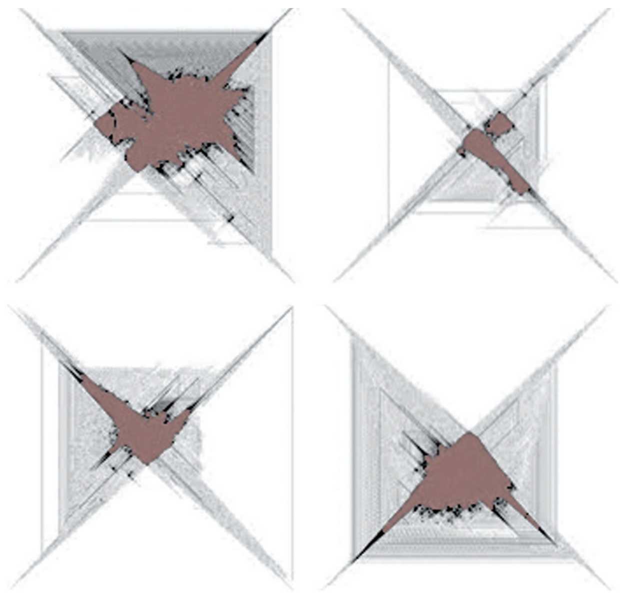


Illustration from the SketchPad manual

Cellular Automata MW2011

A Cellular Automaton is a collection of 'colored' cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells. The rules are then applied iteratively for as many time steps as desired. Von Neumann was one of the first people to consider such a model, and incorporated a cellular model into his 'universal constructor'.



An example of a cellular automaton that has run for 200.000 iterations. These cellular automata ran on the 345/3/6 rule. Source: www.thewildca.com

Cloud, The Gardner

The Cloud is a metaphor for the Internet in cloud computing, based on how it is depicted in computer network diagrams and as an abstraction for the complex infrastructure it conceals. It suggests an omnipresent network that is (wirelessly) accessible everywhere and provides resources and applications to users. Underpinning the notion of the Cloud is that the lion's share of computing power and thus also all computer applications can be delegated to a powerful distributed computer network, i.e., the Internet. All one needs is a light terminal-like device which provides access to the Cloud. This means your computer only needs an operating system and a web browser. Google Chromebook is the first commercial devise based on this principle (released 15 June 2011).

Correlationism Gardner

The internet of things is more than just a technological term: it is a worldview, a framework for understanding the world at large. But it is also a philosophical framework, one that has fallen from grace since Kant. Kant's *Critique of Pure*

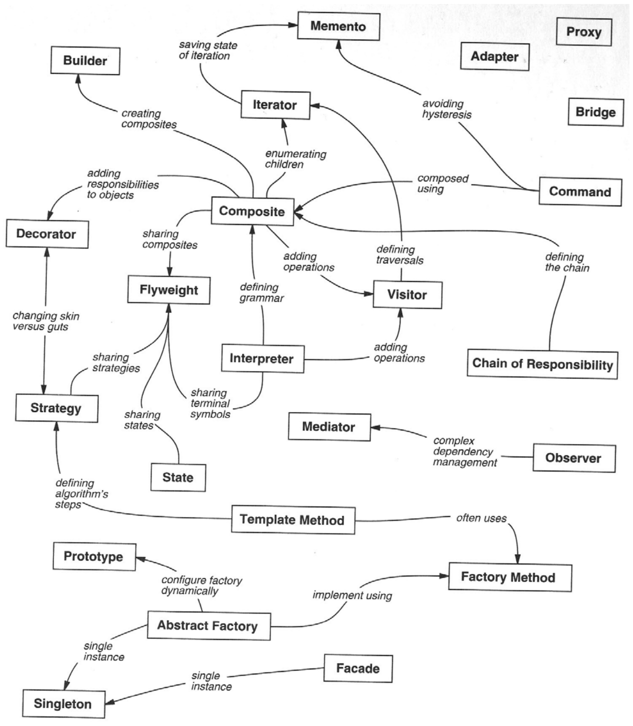
Reason removed philosophical speculation on what the world is and thinking about the nature of reality beyond human reach. Basically any form of realism was rendered naive and uncritical. The consequence of Kant's argument is that the human mind is confined to a transparent cage; all access to reality, all apprehension of the world becomes the world 'for-us' in contrast to the world 'in-itself' – a world to which we have no access to because we cannot have access to the world outside our human point of view and outside language. We are captured in a correlate, the correlate of human-world, subject-object, thinking-being: we cannot step outside. Quintin Meillassoux calls this notion correlationism: *"the idea according to which we only ever have access to the correlation between thinking and being, and never to either term considered apart from the other"* **QM2008**. In recent years a movement has sprung up around the critique of correlationism dubbed 'Speculative Realism.' Although disparate in their approaches, they all try to escape the correlate that has dominated post-Kantian philosophy. One of these is Object-Oriented Philosophy.

Cryptocurrency **WP2011**

Bitcoin is a digital currency created in 2009 by Satoshi Nakamoto. The name also refers to the open-source software he designed that uses it, and the peer-to-peer network that it forms. Unlike most currencies, bitcoin does not rely on trusting any central issuer. Bitcoin uses a distributed database spread across nodes of a peer-to-peer network to journal transactions, and uses cryptography in order to provide basic security functions, such as ensuring that bitcoins can only be spent by the person who owns them, and never more han once.

Design Patterns

In software engineering a design pattern is not a finished design that can be transformed directly into code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are based on Christopher Alexander's architectural theory of patterns, and pattern language. In 1987 Kent Beck and Ward Cunningham began experimenting with the idea of applying patterns to programming and presented their results at the OOPSLA conference that year:
"We propose a radical shift in the burden of design and implementation, using concepts adapted from the work of Christopher Alexander, an architect and founder of the Center for Environmental Structures. Alexander proposes homes and offices be designed and built by their eventual occupants. These people, he reasons,know best their requirements for a particular structure. We agree, and make the same argument for computer programs. Computer users should write their own programs. The idea sounds foolish when one considers the size and complexity of both buildings and programs, and the years of training for the design professions. Yet Alexander offers a convincing scenario. It revolves around a concept called a 'pattern language'". **KBWC1987**



The figure shows the relations between all the software design patterns from Design Patterns (DP1994)

In the following years Beck, Cunningham and others followed up on this work. Design patterns gained popularity in computer science after the book *Design Patterns: Elements of Reusable Object-Oriented Software* was published in 1994 by the so-called 'Gang of Four'. In 1995 Cunningham launched the *Portland Pattern Repository*, an online collection of computer programming design patterns that was accompanied by the *WikiWikiWeb*, the world's first wiki. **WP2011**

Digital Physics **WP2011**

In physics and cosmology, digital physics is a collection of theoretical perspectives based on the premise that the universe is, at heart, describable by information, and is therefore computable. The universe can be conceived as either the output of a computer program or as a vast, digital computation device (or, at least, mathematically isomorphicto such a device). Digital physics is also known as Pancomputationalism or the computational universe theory.

Digital physics is grounded in one or more of the following hypotheses listed here in order of increasing starkness. The universe, or reality, is:

1. essentially informational (although not every informational ontology needs to be digital);
2. essentially computable;
3. essentially digital;
4. itself a computer;
5. in essence the output of a simulated reality exercise.

Konrad Zuse was the first to propose that physics is just computation, suggesting that the history of our universe is being computed on, say, a cellular automaton. His 'Rechnender Raum' (Computing Cosmos / Calculating Space) started the field of Digital Physics in 1967. Today, more than three decades later, his paradigm-shifting ideas are becoming popular. Digital Physics gave rise to Digital Philosophy, a modern re-interpretation of Gottfried Leibniz's monist metaphysics, one that replaces Leibniz's monads with aspects of the theory of cellular automata. Digital Philosophy purports to solve certain hard problems in the philosophy of mind and the philosophy of physics, since, following Leibniz, the mind can be given a computational treatment.

Forrester, Jay

WP2011

Jay Forrester (1918) is a pioneer American computer engineer and systems scientist, and was a professor at the MIT Sloan School of Management. Forrester is known as the founder of System Dynamics which deals with the simulation of interactions between objects in dynamic systems.

Forrester directed the development of DYNAMO (DYNAmic MOdels), a simulation language and accompanying graphical notation developed within the system dynamics analytical framework. It was originally for industrial dynamics but was soon extended to other applications, including population and resource studies and urban planning.

DYNAMO was used for the system dynamics simulations of global resource-depletion reported in the Club of Rome's Limits to Growth.

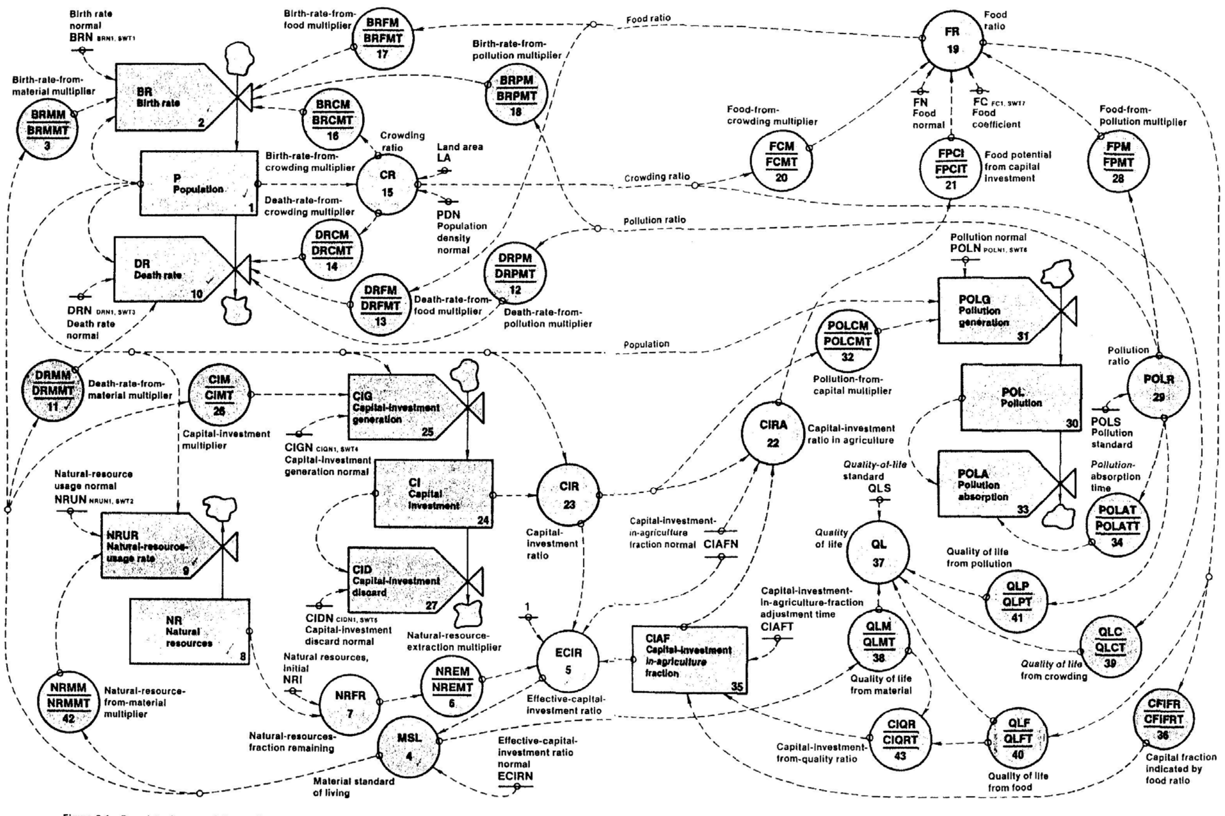
Gardner, Edwin

Edwin Gardner (1980) is an architect, writer and regular contributor to *Volume* magazine. His research and work deals with diagrammatic reasoning in architecture, whether by brain or machine. Currently, he is a researcher at the Jan van Eyck Academie where he is working on his *Diagram Catalogue* project.

Gordian Knots

EG2011

Gordian Knots are hard, if not impossible, problems to solve. They denote the incredible complexity of dealing with or even 'solving' 'certain Gordian Knots problems' in society. Problems can neither be consider in an isolated way, nor can all possible influences and connections be integrated. Nevertheless, one needs to engage the challenge of adding something to the world in a way that is neither naive nor nihilistic. These Gordian knots, when technologized, become Gordian nodes. By technologizing the problem it might seem tractable since technology (especially networks and systems) provide the aesthetics of control, management and rationality.



Birth rate normal BRN
BRFM BRFMT 17
Birth-rate-from-food multiplier
BRPM BRPMT 18
Birth-rate-from-pollution multiplier
BR Birth rate 2
Birth-rate-from-material multiplier
BRMM BRMMT 3
BRCM BRCMT 16
P Population 1
Birth-rate-from-crowding multiplier
Crowding ratio
CR 15
Land area LA
Death-rate-from-crowding multiplier
PDN Population density normal
DRCM DRCMT 14
DR Death rate 10
DRN Death rate normal
DRFM DRFMT 13
Death-rate-from-food multiplier
DRPM DRPMT 12
Death-rate-from-pollution multiplier
Food ratio
FR 19
Food ratio
FN Food normal
FC Food coefficient
Food-from-crowding multiplier
FCM FCMT 20
FPCI FPCIT 21
Food potential from capital investment
Food-from-pollution multiplier
FPM FPMT 28
Crowding ratio
Pollution ratio
Pollution normal POLN
POLG Pollution generation 31
POLCM POLCMT 32
Pollution-from-capital multiplier
Population
DRMM DRMMT 11
Death-rate-from-material multiplier
CIM CIMT 26
Capital-investment multiplier
CIG Capital-investment generation 25
CIGN Capital-investment generation normal
CI Capital investment 24
CIR 23
Capital-investment ratio
CID Capital-investment discard 27
CIDN Capital-investment discard normal
CIRA 22
Capital-investment ratio in agriculture
POL Pollution 30
POLR 29
Pollution ratio
POLS Pollution standard
POLA Pollution absorption 33
Pollution-absorption time
POLAT POLATT 34
Quality-of-life standard QLS
QL 37
Quality of life
Quality of life from pollution
QLP QLPT 41
Natural-resource usage normal NRUN
NRUR Natural-resource-usage rate 9
NR Natural resources 8
Natural resources, initial NRI
NRFR 7
Natural-resource-extraction multiplier
NREM NREMT 6
ECIR 5
Effective-capital-investment ratio
Capital-investment-in-agriculture fraction normal CIAFN
CIAF Capital-investment-in-agriculture fraction 35
Capital-investment-in-agriculture-fraction adjustment time CIAFT
QLM QLMT 38
Quality of life from material
QLC QLCT 39
Quality of life from crowding
NRMM NRMMT 42
Natural-resource-from-material multiplier
Natural-resources-fraction remaining
MSL 4
Material standard of living
Effective-capital-investment ratio normal ECIRN
CIQR CIQRT 43
Capital-investment-from-quality ratio
QLF QLFT 40
Quality of life from food
CFIFR CFIFRT 36
Capital fraction indicated by food ratio

Figure 2-1 Complete diagram of the world model interrelating the five level variables — population, natural

Forrester's diagram of the world model interrelating the five level variables – population, natural resources, capital investment, capital-investment-in-agriculture fraction, and pollution.

Industrial Foundation Classes (IFC) WP2011

Industrial Foundation Classes were introduced to describe building and construction industry data according to specific standards. It is a neutral and open specification that is not controlled by a single vendor or group of vendors. It is an object-based file format with a data model developed by buildingSMART (International Alliance for Interoperability, IAI) to facilitate interoperability in the building industry, and is a commonly used format for Building Information Modeling (BIM). The IFC model specification is open and available. It is registered by ISO as ISO/PAS 16739 and is currently in the process of becoming the official International Standard ISO 16739. Because of its focus on ease of interoperability between software platforms the Danish government has made the use of IFC format(s) compulsory for publicly funded building projects. Finnish state-owned facility management company Senate Properties also requires IFC compatible software and BIM in all their projects.

Information Asymmetry

When one party in an economic transaction has privileged and/or better access to specific knowledge which is inaccessible or hard to attain for the other party, the first party profits from the information asymmetry between buyer and seller of a goods or service; for example, in fields such as legal advice, finance, medicine and education. In economic terms, the condition of information asymmetry is undesirable, whether the lack of information lies with the buyer or seller. While one can say that there is always some degree of information asymmetry in every transaction, too much asymmetry results in suspicion and a decrease in trust between the trading parties. It results in a non-transparent and dysfunctional market, and can even lead to the disappearance of a specific market since there is no exchange of information or mutual trust at all. This is why markets involving specialist knowledge are highly regulated, professionalized, and/or intermediated: in large part to protect the consumer BLP2007. In the relationship between owner/client and contractor these mechanisms are lacking or work very poorly. There is a lot of room for the contractor to take advantage of information asymmetry. An excerpt from a 2007 book by Barry B. LePartner, the director of a law firm solely serving as business and legal advisor to design professionals, illustrates this problem:
"The fact that owners do not have near equal knowledge [as the contractor] is the major reason why inefficiency and mutable costs persist in construction. For a number of reasons peculiar to construction, building owners cannot easily compare building price or quality – at the start, during construction, or even after the job is done. Most owners cannot even read blueprints, much less fathom the complex process of transforming the prints into usable structures. In most instances, the construction team pre-sets the project budget by reviewing the design documents prepared by the architect and the engineers. There is rarely anyone equally knowledgeable about material or labor costs to effectively challenge the budget set by the contractor. Most inexperienced owners cannot readily distinguish between reasonable and unreasonable contractor bids. Even when they can, their only real alternative if the price comes in higher than the project budget is to reduce or eliminate features since the contractor will be unlikely to reduce its overall price without a commensurate scope reduction. Even then, the contractor will maintain the same degree of profitability." BLP2007

LAN Party (Local Area Network Party) WP2011

A LAN Party is a temporary, sometimes spontaneous, gathering of people with computers with which they establish a local area network (LAN) primarily for the purpose of playing multiplayer computer games. The size of these networks may vary from very small (two people) to very large. Small parties can form spontaneously, but large ones usually require a fair amount of planning and preparation.

LAN party events differ significantly from LAN gaming centers and Internet cafés in that they generally require attendants to bring their own computer (BYOC) and are not permanent, often taking place in general meeting places or residences.

LAN parties have their own unique culture. Case modding (modifying a computer case in any non-standard way) enthusiasts often show off computers with flashy aftermarket lighting, LCD screens, enhanced speakers, and many other computer accessories. Highly caffeinated drinks, termed energy drinks, are very popular at these events to improve concentration and stamina as LAN parties often run into the early morning hours. Large parties can last for several days with no scheduled breaks. Participants often play through the night and into the next day although there is often a designated room separated from the LAN party in which to sleep. BIMStorm could be qualified as a 24-hour LAN Party of sorts for architect, engineers and building consultants, although technically is it uses the Internet, not the LAN.

Latour Litany FG2010

Ian Bogost coined the term Latour litany, referring to Bruno Latour's Actor-Network-Theory. A 'Latour litany' is any list of objects/actors in the world designed to provide an expressionist sample of the lavishness of the non-human world. For example 'washing machines, snowstorms, blades of grass, satellites, gods, pots, paintings, laws, horseshoes and engines' is a Latour litany.

Lingua Franca WP2011

Lingua Franca (also working language, bridge language, or vehicular language) is a language systematically used to make communication possible between people not sharing a mother tongue, in particular when it is a third language distinct from both mother tongues. The equivalent of a lingua franca in software are open standards, which provide the

possibility for data to be produced, exchanged and read by anyone. A good example of this are Extensible Markup Language (XML) based formats. The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for every language. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example, in web services. Many application programming interfaces (APIs) have been developed which software developers use to process XML data, and several schema systems exist to aid in the definition of XML-based languages.

Examples of XML formats available in the building industry are ifcXML (Industrial Foundation Classes XML), gbXML (Green Building XML), and aecXML (Architecture, Engineering and Construction XML). Within the building industry the Industrial Foundation Classes are an effort to realize these open standards.

Lock-in WP2011

Lock-in can result from network effects. For example, if network standards are open enabling competitive implementation by different vendors, there is no vendor lock-in. One example of this would be email, for it has a considerable network effect but there is interoperability between different email providers, thus no lock-in. Apple's App Store has had an enormous network effect because of the popularity of the iPhone, but one has to comply with Apple's rules to be able to use the network; those in the network are locked-in.

Machine Readable WP2011

In telecommunications a machine-readable medium (automated data medium) is a medium capable of storing data in a machine-readable format that can be accessed by an automated sensing device and be turned into (practically in every case) a binary form. Examples of machine-readable media include magnetic disks, cards, tapes, and drums, punched cards and paper tapes, optical disks, barcodes and magnetic ink characters. Common machine-readable data storage and data transmission technologies include processing waveforms, optical character recognition (OCR), and barcodes. Any information retrievable by any form of energy can be machine-readable.

Mars, Marcell

Marcell Mars aka Nenad Romic (1972) is a free software advocate, cultural explorer and social instigator. Mars is one of the founders of Multimedia Institute – mi2 and net.culture club mama, both located in Zagreb. He initiated the GNU GPL publishing label EGOBOO.bits, started Skill sharing meetings of enthusiasts at mama and Skill sharing's satellite events – g33koskop, 'Nothing will happen' and 'The Fair of Mean Equipment'. When in Zagreb Marcell hangs out in Hacklab at mama; in Belgrade he runs Wonder of technology/Cudo tehnike at the Faculty of Media and Communication; and at the Jan van Eyck Academie in Maastricht he is working on the project Ruling Class Studies and collaborates with Edwin Gardner on Tracing Concepts. also sings, dances, tells stories and makes music as Nenad Romic za Novyi Byte.

Material-semiotic WP2011

Material-semiotic refers to a form of analysis that maps relations that are simultaneously material (between things) and semiotic (between concepts). It assumes that many relations are both material and semiotic. For example, the interactions in a school involve children, teachers, their ideas, and technologies (such as tables, chairs, computers and stationery). Together, these form a single network.

Meta-circular Evaluator Mars

An interpreter in computer science is a computer program which executes code written in some programming language. It is different from a compiler which translates code written in some programming language into a stand-alone, executable program. A self-interpreter, or meta-interpreter, is a programming language interpreter written in the language it interprets. A special case of self-interpreter is meta-circular interpreter which can accept input code written in the programming language in which it itself is written without any modification. The input code can be accepted without any modification because of homoiconicity, a property of programming language in which the primary representation of programs is also a data structure in a primitive type of the language itself (*"code is data, data is code…"*).

Network Effect WP2011

In economics and business, a network effect (also called network externality or demand-side economies of scale) is the effect one user of a good or service has on the value of that product to other people. When network effect is present, the value of a product or service increases as more people use it.

The classic example is the telephone. The more people own telephones, the more valuable the telephone is to each owner. This creates a positive externality because a user may purchase a telephone without intending to create value for other users, but does so in any case. Online social networks work in the same way, with sites like Twitter and Facebook being more useful the more users join.

The expression 'network effect' is applied most commonly to positive network externalities as in the case of the telephone. Negative network externalities can also occur when more users make a product less valuable, but are more commonly referred to as 'congestion' (as in traffic congestion or network congestion).

Over time positive network effects can create a bandwagon effect as the network becomes more valuable and more people join in a positive feedback loop.

Neumann, John von WP2011

John von Neumann (1903 – 1957). A Hungarian-American mathematician and a pioneer of the application of operator theory to quantum mechanics as well as in the development of functional analysis. He was a principal member of the Manhattan Project and the Institute for Advanced Study in Princeton (as one of the few originally appointed), and a key figure in the development of game theory and the concepts of cellular automata, the universal constructor, and the digital computer.

Object-oriented Gardner

Hello, Everything*

As we speak the web is spilling into the physical world. The internet is growing into things, into objects, into literally everything. The internet is like a brain in a vat that is now rapidly extending out of its dark sensory-deprivation tank. Growing neurons and limbs, developing more senses than any human ever had, and carefully feeling its way around the great outdoors. Today, most visions of an internet of things focus on a particular rendering of urban life. They focus on the interaction between city and user as an extension of the personal computing paradigm distributed over many devices, whether static or mobile, whether controlled with gestures or by touch. This vision privileges the user's point of view on the city. It does not register the real implications, not to mention the real disruptions, of weaving the internet into the built environment. The anthropocentric point of view prevents us from seeing other perspectives than that of the user. It creates a blind spot for other concerns and other perspectives, such as those of things – the entities that populate the non-human world. This is precisely what we need to consider: 'the things themselves'. It's not about writing off the human perspective; it's about having an additional perspective, to see the human as equal to things, as an object amongst objects, a flat hierarchy, a democracy of objects. The change of view from user-oriented to object-oriented is necessary when one wants to consider and work with the idea that the city is an internet of things: in other words, a collection of bricks, cars, glass, steel, piping, wires, air conditioning, streets, interiors, furniture, permits, a city hall, a cathedral, construction sites, zoning laws, window washers, fraternities, garbage collecting, policing, squatting, sewer systems, traffic lights, cars, billboards, sidewalks, squares, parks, apartments, penthouses, junkies, prostitutes, fathers, sons, hospitals, metro lines, bike stands, nuts and bolts…one could continue this Latour litany forever. The internet of things is the world of real and virtual objects. Each object can have behaviors, characteristics, internal workings, external affects, particular methods or practices. Each object relates to other objects by hierarchy, affiliation, set, or sequence. Each object can mobilize other objects, move in clusters and swarms, reinforce their constellation and gain meaning and influence. This world view is classified as 'object-oriented' or as 'material-semiotic' webs or networks. Fields are springing up around these world views like object-oriented philosophy in terms of theorizing, object-oriented programming in terms of operating and Actor-Network-Theory in terms of analyzing. The object-oriented view is a worldview, a perspective unbound by disciplinary perimeters; it is a worldview disseminated throughout society wherever software and computing emerges.

Language is the platform upon which the naming and relating of objects into a fabric of meaning occurs. With the internet of things, language is no longer about us conversing about objects, it's us conversing with objects, and furthermore objects conversing with each other about us. This is what differentiates natural language from programming language (or even more fundamentally from 'computation' see digital physics). Programming language is not about the world anymore. It's not language as transparent cage standing between the human-thinking subject and being-in-the-world (the world 'out there', the world of objects); it's as if lingual objects can speak the same language as the sentences in which they are found. It's like a text that can talk to itself, rewrite itself, and write other new texts with new objects. This is what it would mean when objects are increasingly woven into a ubiquitous technological fabric. Objects can sense things, give them and themselves names; they can speak and write about things. But these object do not have a free will – they are determined but still unpredictable. Metaphor and ambiguity are alien concepts to this language, but this genus of language does have the some of the same characteristics of natural languages, most importantly that language can exclude or include. Of key importance for the internet of things and its potential is to have a mutual agreement on vocabulary and grammar – a lingua franca for objects.

A network of things would be relatively impotent if these things did not adopt a common language. Things that could exclusively communicate with similar things would ultimately be locked-in. Whether this is beneficial or not depends on how power in the network is organized and exercised, and on what side of the power relation objects are. When certain knowledge is inaccessible for some who need it, those who do have access can benefit; in economic terms this is known as information asymmetry. Information asymmetry is the single biggest obstacle to be overcome to truly network the built environment – to allow contractors, realtors, homeowners, architects, city authorities, engineers and project developers to cooperate openly on a level informational playing field. Only at this point can the built environment really reach network effect and take full advantage of the potential of the Building Code.

* The title comes from the 'Hello, Everything' conference on speculative realism and object-oriented ontology held at UCLA, December 1, 2010.

Object-Oriented Programming (OOP) WWW2011, WP2011

Object-Oriented programming is a programming language paradigm using 'objects' – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. An object-oriented system, language, or environment should include at least the following properties: Encapsulation, meaning restricting access to some of the object's components. Polymorphism, meaning the ability to create a variable, a function, or an object that has more than one form. And inheritance, meaning a way to compartmentalize and reuse code by creating collections of attributes and behaviors called objects which can be based on previously created objects.

Polymorphism and Inheritance are certainly patterns that facilitate OO programming, but encapsulation seems to be the key distinction between OO and procedural programming: *"asking data to do things instead of doing things to data"* (David Wright).

Object-Oriented Philosophy WP2011

The central tenet of object-oriented philosophy (OOP) is that objects have been given short shrift for too long in philosophy in favor of more 'radical approaches.' Graham Harman has classified these forms of 'radical philosophy' as those that either try to 'undermine' objects by saying that objects are simply superficial crusts of a deeper, underlying reality either in the form of monism or a perpetual flux, or those that try to 'overmine' objects by saying that the idea of a whole object is a form of folk ontology, that there is no underlying 'object' beneath either the qualities (e.g., there is no 'apple', only 'red', 'hard', etc.) or the relations (as in both Latour and Whitehead, the former claiming that an object is only what it *"modifies, transforms, perturbs, or creates"*). OOP is notable for not only its critique of forms of anti-realism, but other forms of realism as well. Harman has even claimed that the term 'realism' will soon no longer be a relevant distinction within philosophy as the factions within Speculative Realism grow in number. As such, he has already written pieces differentiating his own OOP from other forms of realism which he claims are not realist enough as they reject objects as *"useless fictions"*.

Object oriented ontology has also found favor among a group of philosophers including Levi R. Bryant, Ian Bogost, and Timothy Morton, who blog often and are preparing to publish several full-length monographs on their varying ontologies. Bryant's volume will be titled 'The Democracy of Objects'; Bogost's will be 'Alien Phenomenology'; and currently Morton is working on a volume called 'Dark Ecology', which promises to explicate his idea of hyperobjects.

Related: Correlationism

OOPSLA CA1996

The Object-Oriented Programs, Languages and Applications conference wittnessed Christopher Alexander as Keynote. What follows is the closing fragment of his lecture in a hall full of computer activists and software engineers in San Jose, California in 1998.

"When a paradigm change occurs in a discipline it is not always the members of the old profession who take it to the next stage. In the history of the development in technical change very often the people responsible for certain specialty are then followed by a technical innovation. And then the people who become responsible for the field after the technical innovation are a completely different group of people. When the automobile came along the people who built the buggies for the horse and buggy did not then turn into Henry Ford. Henry Ford knew nothing about horse buggies. The people who were building automobiles came from left field and then took over – and the horse and buggy died off.

It is conceivable to imagine a future in which this problem of generating the living structure in the world is something that you – computer scientists – might explicitly recognize as part of your responsibility. (…) The idea of generative process is natural to you. It forms the core of the computer science field. The methods that you have at your fingertips and deal with everyday in the normal course of software design are perfectly designed to do this. So, if only you have the interest you do have the capacity and you do have the means. (…) What I am proposing here is something a little bit different from that. It is a view of programming as the natural genetic infrastructure of a living world which you/we are capable of creating, managing, making available, and which could then have the result that a living structure in our towns, houses, work places, cities, becomes an attainable thing. That would be remarkable. It would turn the world around, and make living structure the norm once again, throughout society, and make the world worth living in again.

This is an extraordinary vision of the future in which computers play a fundamental role in making the world – and above all the built structure of the world – alive, humane,ecologically profound, and with a deep living structure".

Pancomputationalism

see Digital Physics

Pattern

From things to patterns, Christopher Alexander explains: *"It is one thing to say in a kitchen, for example, you have a certain relationship between a counter, a refrigerator, a sink, and a stove. Everyone can see that. But in that view of the thing you still consider the kitchen to be made of the counter, refrigerator, sink, and stove, and their relationship is kind of playing a secondary role in trying to organize it. But when you look more closely you realize that the stove is a relationship between an oven, some heaters, and some switches and furthermore, that the switch is a relationship between something you can turn with your hand and some electrical contacts, and so on. Finally you realize that the whole substance of all this is in fact made of these patterns and that the 'things' are just convenient labels which we give to bundles of patterns, or patterns themselves."* SG1983
Reyner Banham adds: *"The heart of Alexander's matter is the concept of a 'pattern', which is a sort of package of ideas and forms which can be subsumed under a label as commonplace as 'comfortable window-seat' or 'threshold' or 'light on two sides of a room', or as abstract as 'intimacy gradient'. Such a labeled pattern contains not only the knowledge of the form and how to make it, but 'there is an*

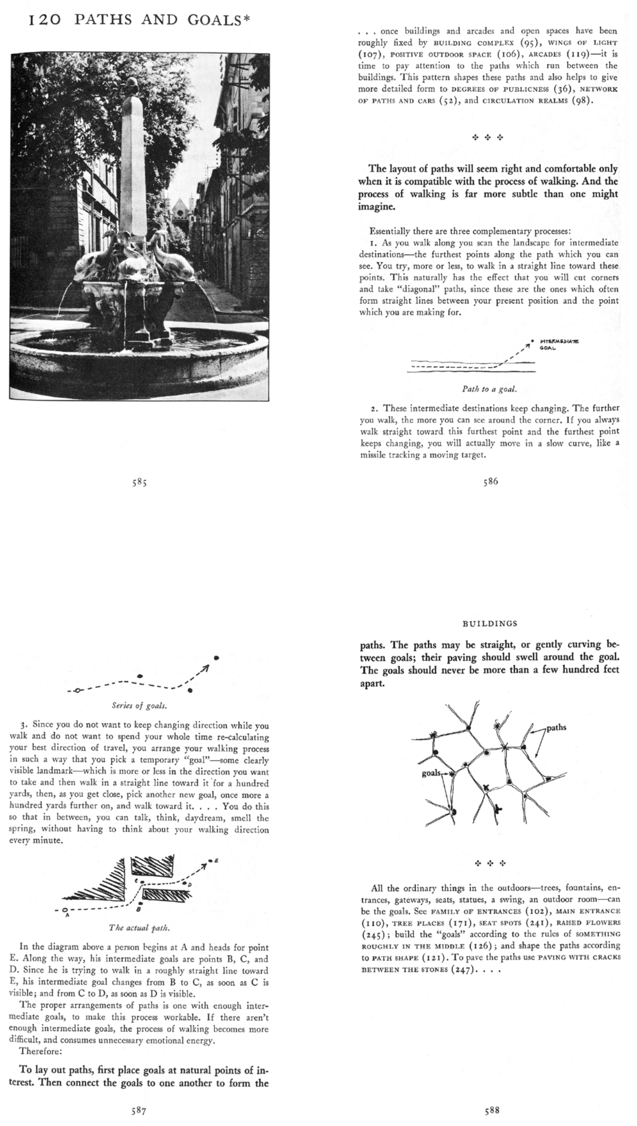
imperative aspect to the pattern … it is a desirable pattern… [the architect] must create this pattern in order to maintain a stable and healthy world.' (…) In other words, each pattern will have moral force, will be the only right way of doing that particular piece of designing – at least in the eyes of those who have been correctly socialized into the profession. (…) And in general, as an outsider who was never socialized in the tribal long-house, it seems to me that Alexander's patterns are very like the kind of packages in which architects can often be seen to be doing their thinking, particularly at the sort of second sketch stage when they are re-using some of what was sketched out in the first version." RB1990

Pattern Language

A Pattern Language is the approximation of *"the idea that a set of rules could actually generate a building is as disturbing as the idea that a human being is generated by a few genetic rules operating on chromosomes or that a poem is generated by a few grammatical rules operating on language. [This] is precisely what Alexander is claiming. For him, the two examples just cited – genetics and linguistics – are not just analogies. In each case there is a principle of 'generativity' involved and Alexander is not just interested in a theoretical equivalent of this principle. He is actually interested in generativity itself and is therefore serious about a set of rules which generates buildings as structural principle of natural creation as it is understood in modern science."* SG1983
Pattern Language is a concept developed by Christopher Alexander as a theory for the structure of our built environment. *"The elements of this language are entities called patterns. Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice."* CA1977.

The theory was intended to eventually become a genuinely generative approach, something which never happened. But the format of the *pattern language* is a powerful idea that has been adopted across various disciplines, most prominently in object-oriented programming in the design patterns movement. It might very well be that Alexander's Pattern Language might somehow end up back in the built environment again via the field of programming and the Internet of things – a premonition one might have experienced when Alexander gave his keynote address to a lecture hall full of computer scientists and software engineers in October 1996 at the Association for Computing Machinery (ACM) Conference on Object-Oriented Programs, Systems, Languages and Applications (OOPSLA) – San Jose, California.

Within architecture Alexander's theories have fallen from grace. This has more to do with Alexander's unnuanced and unforgiving tone against practically anything modern, contemporary and artistic, and advocating a pre-industrial, traditionalist and romantic view with absolutist ideas about beauty. Although Alexander's tone and his theory have obvious shortcomings, he has been successful in grasping some of architecture's essence. Reyner Banham explains:


120 PATHS AND GOALS*

585

. . . once buildings and arcades and open spaces have been roughly fixed by BUILDING COMPLEX (95), WINGS OF LIGHT (107), POSITIVE OUTDOOR SPACE (106), ARCADES (119)—it is time to pay attention to the paths which run between the buildings. This pattern shapes these paths and also helps to give more detailed form to DEGREES OF PUBLICNESS (36), NETWORK OF PATHS AND CARS (52), and CIRCULATION REALMS (98).

✣ ✣ ✣

The layout of paths will seem right and comfortable only when it is compatible with the process of walking. And the process of walking is far more subtle than one might imagine.

Essentially there are three complementary processes:

1. As you walk along you scan the landscape for intermediate destinations—the furthest points along the path which you can see. You try, more or less, to walk in a straight line toward these points. This naturally has the effect that you will cut corners and take "diagonal" paths, since these are the ones which often form straight lines between your present position and the point which you are making for.

Path to a goal.

2. These intermediate destinations keep changing. The further you walk, the more you can see around the corner. If you always walk straight toward this furthest point and the furthest point keeps changing, you will actually move in a slow curve, like a missile tracking a moving target.

586

Series of goals.

3. Since you do not want to keep changing direction while you walk and do not want to spend your whole time re-calculating your best direction of travel, you arrange your walking process in such a way that you pick a temporary "goal"—some clearly visible landmark—which is more or less in the direction you want to take and then walk in a straight line toward it for a hundred yards, then, as you get close, pick another new goal, once more a hundred yards further on, and walk toward it. . . . You do this so that in between, you can talk, think, daydream, smell the spring, without having to think about your walking direction every minute.

The actual path.

In the diagram above a person begins at A and heads for point E. Along the way, his intermediate goals are points B, C, and D. Since he is trying to walk in a roughly straight line toward E, his intermediate goal changes from B to C, as soon as C is visible; and from C to D, as soon as D is visible.

The proper arrangements of paths is one with enough intermediate goals, to make this process workable. If there aren't enough intermediate goals, the process of walking becomes more difficult, and consumes unnecessary emotional energy.

Therefore:

To lay out paths, first place goals at natural points of interest. Then connect the goals to one another to form the

587

BUILDINGS

paths. The paths may be straight, or gently curving between goals; their paving should swell around the goal. The goals should never be more than a few hundred feet apart.

✣ ✣ ✣

All the ordinary things in the outdoors—trees, fountains, entrances, gateways, seats, statues, a swing, an outdoor room—can be the goals. See FAMILY OF ENTRANCES (102), MAIN ENTRANCE (110), TREE PLACES (171), SEAT SPOTS (241), RAISED FLOWERS (245); build the "goals" according to the rules of SOMETHING ROUGHLY IN THE MIDDLE (126); and shape the paths according to PATH SHAPE (121). To pave the paths use PAVING WITH CRACKS BETWEEN THE STONES (247). . . .

588

"Looking back on the early days of his 'pattern language', [Alexander] revealed one of its apparent failures to his biographer, Stephen Grabow:

"Bootleg copies of the pattern language were floating up and down the west coast, and people would come and show me the projects they had done, and I began to be more and more amazed that, although it worked, all these projects looked like any other buildings of our time (…) still belonged perfectly within the canons of mid-century architecture."

Now, if one hoped that the pattern language would be a revolutionary way of designing buildings, a new paradigm in architecture comparable with the Copernican revolution in cosmology, then clearly the project had failed and further research was indeed needed. But, in another light, the failure of the pattern language to change the nature of architectural design could be seen as something of a triumph: an unwitting first approximation description of what architects actually do when they do architecture.

(…) Such patterns – perhaps even a finite set of patterns – and their imperatives seem to be shared by all architects. … This is not to say that Alexander's accidental revelation exhausts the topic, far from it; for a start, it is still much too crude to explain anything really subtle. Being cast in a prescriptive, rather than a descriptive, format, it avoids such questions as how such patterns are formed, and where, and cannot support the kind of anthropological investigation that has revealed the workings of other secret cultures to us in the past." RB1990

Program NE2010

"Program has come to mean any prearranged information that guides subsequent behavior, as in, for example: formal proceedings (1837), political platforms (1895), broadcast presentations (1923), electronic signals (1935), computer instructions (1945), educational procedures (1950), and training (1963)."

Programming Mars

Evolutionary Stable Concepts and Strategies

A Short History of Making Machines Work

In a period of roughly sixty years, from military-funded mainframes with privileged access to the personal computer of today, the concepts that sprung out from the Universal Machine have seen great evolution. Some have failed, while others have become stable concepts within the field. In game theory, a strategy adopted by a population of players cannot be invaded by any alternative strategy that is initially rare – this kind of strategy is denoted as *evolutionary stable*. Borrowing from this, here we consider ourselves with the evolutionary stable concepts and strategies in computing, where the 'players' are the computer boys, and the concepts and strategies they adhere to constitute their culture.

Any particular computer's spectrum of usefulness is limited by its performance. In the earliest computers, every bit (one or zero) had to be entered, one at a time, on rows of mechanical switches or by jumper wires on plug boards in order to make them perform a specific task. It would take ages to write even the simplest of programs. How to automate these repetitive tasks was an obsession for programmers from the very start of software development. Often, the automatization would occur by writing programs which simulated commonly repeating patterns found in a variety of tasks. However, one pays a price for bringing in programming expressiveness and abstractions that are more understandable to humans than machine code. The computer has to do the extra work of converting lines of code written in programming language into machine-executable sequences of bits. So in order to write in code, a language that is more accessible for humans than `01001001110101`, one has to sacrifice machine performance – an abstraction penalty, as programmers call it.

In 1952 Grace Hopper, a pioneer in the field of computer science, and her team developed a system called the 'A-0 Compiler' which could translate one line of code written by a programmer into several machine instructions (bits). The compiler was invented to improve the productivity of programmers by letting them express their solutions in more human-readable language: programming language. It took five more years for hardware to catch up and become fast enough for that process to be efficient. In April 1957, after two and a half years of work, a team of a dozen programmers led by John Backus released a programming language called FORTRAN (IBM Mathematical Formula Translating System) – and subsequently became a huge success. General Motors estimated that the productivity of programmers had been increased by a factor somewhere between 5 and 10, and that, even taking into account the extra time the machine needed to run the compiler program, the overall cost of programming was reduced by a factor of 2.5. MCK2003

Grace Hopper's team didn't sit silently. A couple years later, in 1959, together with other researchers from private industry, universities, and government, they created a committee responsible for what would become one of the most successful programming languages of all time: COBOL (COmmon Business-Oriented Language). While FORTRAN became the programming language of choice for science and mathematics, COBOL mainly worked for business and administration.

During the fifties a number of research activities established a basic suite of programs and terminology. In other words, terms such as programming language, compiler, utility, and mathematical subroutine became stable concepts. This was also the period in which the organizational model of how the proposed technology was to be developed was established – a stable strategy, one could say.

This model was one of cooperative association, where skill and knowledge were being shared among a group of developers in a forum-like setting.

But it was not solely developers who were advancing the field. In November 1952 the Digital Computer Association was founded, an owner- and user-group of IBM 701s. They often held informal meetings, one outcome of which became PACT, a new programming system that was considerably better than anything IBM was offering at the time. Next, the users of the following IBM model, the 704, founded SHARE. Their organizational model – distributed development of a library of programs, cooperation, informal communication, and discussion groups – remains the preeminent way for computer users to interact with one another to the present day; furthermore it has served as inspiration for hobbyists of the Homebrew Computer Club in the 1970s, as well as the Free software movement since the 1980s. The SHARE user group developed the world's first operating system: a program which lets other programs run by taking care of the common management tasks of hardware resources. The operating system would then take over a role that was once fulfilled by bare hardware. From that point on most software applications were developed to run on top of an operating system, instead of on bare hardware. Importantly, this was the first developmental step in making the machine increasingly virtual.

It didn't take too long after that – 1967, to be exact – for hardware performance to improve enough to implement an operating system with two components: Control Program

(CP) and Console Monitor System (CMS). CP first creates a virtual machine environment in which a standalone mainframe computer (IBM System/360) is simulated. On top of that simulated computer, a lightweight, single-user operating system could be run – the Console Monitor System (CMS). The CP/CMS made it possible to virtually divide a large mainframe into smaller mainframe computers each of which could then be independently and simultaneously used (known as time-sharing). So the very same hardware (IBM System/360/67) on which software is running is simulated in as many virtual instances as computational power would allow.

While computational power was constantly increasing, the physical space it required was steadily decreasing. Room-sized mainframes were shrinking into mini-computers that could fit beneath a desk. With its small size, its keyboard as a handy input device, a feeling of individual usability through time-sharing access, and the sensation of interactivity, the mini-computer was becoming much easier to use. It opened up the imagination to the possibility of giving the general public direct access to computers.

It was in the San Francisco Bay area during the late 1960s and early 1970s that this idea found especially fertile ground. The passionate idea arose that computers could become vehicles of empowerment and freedom, of radical personal and social transformation. These ideas were rapidly multiplying in the minds of those working in an area of just a few blocks; situated here were the offices of Stewart Brand's Whole Earth Catalog, Douglas Engelbart's Augmentation Research Center (ARC), and the People's Computer Company (later Homebrew Computer Club). **FT2006**

On December 9, 1968 Stewart Brand filmed Engelbart's the 'mother of all demos' to an audience of a thousand computer professionals and amateurs. It showed an experimental computer system in which mouse-keyboard-screen, video conferencing, teleconferencing, email, hypertext, word processing, hypermedia, object addressing and dynamic file linking, bootstrapping, and a collaborative real-time WYSIWYG editor (WhatYouSeeIsWhatYouGet) were all introduced. In attendance, as well, were two members of the Homebrew Computer Club: Steve Wozniak and Steve Jobs.

Nine years later, heavily inspired by this 'mother of all demos,' Wozniak and Jobs launched Apple II. This started the real revolution of the personal computer. As a result, a strange variety of the computer boy breed began to emerge – one who could now work on his own, outside of the large, centralized mainframes guarded over by institutions. Sure enough, the personal computing movement garnered its own heroes, or rather, antiheroes: hackers. A hack was defined as *"a project undertaken or a product built not solely to fulfill some constructive goal, but with some wild pleasure taken in mere involvement"* **SL1984**.

The hackers would come from MIT Artificial Intelligence Lab, Homebrew Computer Club and the 'New Age' scene found at Whole Earth Catalog. From their keyboards and screens they shared a common ethos, a 'hacker ethic' stating:

- Access to computers – and anything which might teach you something about the way the world works – should be unlimited and total. Always yield to the Hands-On Imperative! …
- All information should be free …
- Mistrust Authority
- Promote Decentralization …
- Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race, or position …
- You can create art and beauty on a computer …
- Computers can change your life for the better.

SL1984

This hands-on implementation of the idea that 'all information should be free' confronted the very basis of many industries, and would continue to do so for decades to come. But the hacker ethic is also today still very much present in the corporate world of software development. Conventionally, one of the criteria that establishes a profession is a formalized procedure of obtaining a title. But innovative, creative and excellent programmers may not hold even a programming degree or perhaps have no formal education at all. Criticizing any form of discipline of rigid formalism, knowledge between hackers is transmitted, peer-to-peer, during gatherings at informal meetings (known as unconferences) or during endless discussions in chatrooms that run on protocols from the late 80s. Large software empires celebrate the myths of their foundations by young geniuses who dropped out of university out of boredom. One of the most reputable cult software companies today, GitHub, takes pride in the fact that half of the programmers never obtained a degree. **OM2011**

The counter-culture revolution provides the computing field with a powerful set of concepts and strategies that have thus radicalized the cooperative gatherings of mainframe programmers into anarchic meetings of hackers and amateurs with a mission to change the world with the help of PCs into a self-organizing and information-sharing techno-utopia.

Returning to the issue of technology, the virtual machine, a self-contained operating environment that behaves as if it is a separate computer, had proved to be a very useful environment for software development. With a virtual machine one had an environment within which one could perfect a programming language that could be closer to the principles of the Universal Machine – an environment that ideally would make programs completely platform-independent.

When the Macintosh, the first successful personal computer with mouse and Graphical User Interface (GUI), was released this shifted programmers' priorities towards concerns of education and ease of use. Until the late 1970s the typical computer system had to be operated like a factory assembly line: a batch processing system received input data that typically had the format of sequential file on tape, that file was processed by a program, the results were written into another sequential file that in turn was again processed by another program, and so on. The procedural model of computation was not the most suitable for the GUI system. The GUI is nothing like a factory; it operates more like a stimulus/response machine. The system quietly waits for a stimulus, or event, and when the event arrives the system springs to life, processing the event before returning to a waiting state. Based upon this idea Alan Kay and Dan Ingalls developed the Smalltalk programming language at Xerox PARC in the late seventies and early eighties. Smalltalk was conceptualized as an object-oriented language that would provide an ultimate stimulus/response human-computer symbiosis. In the world of Smalltalk everything is an object. Smalltalk objects are in a certain state of being which can be queried or changed by messages from other objects or by the object itself. While the Smalltalk virtual machine is running all of the objects, each object can observe and modify its own (as well as the overall) structure and behavior (of all the other objects in the system). Smalltalk is a totally reflective system which means that it can modify its own structure and behavior while it is running. `smalltalk := 'a lot of small talks'` Got it? Beautiful. **SF2006**

Together with LISP, Smalltalk was trying to attain a state of perfection within its own universe – its own self-referential system. It was the beauty of reinventing the whole history of computing within the boundaries of one programming language. But this beauty comes as a price: namely, that every communication with the world outside that self-referential universe makes the code messier, less pure. Even communication between LISP and Smalltalk – deemed two of the most advanced and beautiful programming languages amongst programmers – pollutes the purity of their respective universes because one of them (Smalltalk) doesn't have a characteristic called homoiconicity (*"code is data, data is code…"*). Homoiconicity is a property of some programming languages in which the primary representation of programs is also a data structure in a primitive type of the language itself, which means you can write programs which write programs that can modify themselves while running, This operation is known to programmers as a meta-circular evaluator. While this characteristic is an integral part of LISP, Smalltalk can do this only through a workaround without the homoiconic property thereby resulting in contaminated universes. Beside the perfect universes LISP and Smalltalk were designed to be, they were also parts of very ambitious projects. LISP became the favored programming language in Artificial Intelligence research, while Smalltalk was inspired by Seymour Papert's constructionist learning theories and was to be the ultimate environment for learning programming.

Both projects failed. They failed in the world where progress is iterative, one small concept per cycle of understanding; moreover, these languages were proprietary and not open for the community for continued development. LISP and Smalltalk failed at a time when personal computers were neither sufficiently powerful, widely disseminated, or fully interconnected. Although they are less in use today, the beauty of LISP and Smalltalk have become influential touchstones in the world of programming. And maybe they will one day come back with a bang!

The Internet naturally changed everything – except the good old habit of running brand-new virtual machines on top of other virtual machines or operating systems. Web browsers started to become the most used, truly multiplatform application. The object-oriented paradigm in programming reached its peak with the release of the canonical book, *Design Patterns* **DP1994**. In 1995 Sun Microsystems launched the Java programming language and virtual machine; and Java was also object-oriented. It promised to *"Compile Once, Run Everywhere"*, meaning developers could expect the Java Virtual Machine to run on all possible platforms – in operating systems, in computer architectures, and inside every web browser – making the Java programming language truly platform independent.

Although Java failed to fulfill the promise of becoming a major software development environment for desktop application or small web apps, it definitively became the industry standard in the field of server enterprise technologies and, recently, became the heart of the most popular operating system for smartphones – Google's Android.

The Universal Machine – and the concept of the universe being a cellular automaton – are beautiful ideas that have already inspired a couple of generations of programmers. Software developers cultivate an ecosystem in which these ideas circulate, evolve, thrive, succeed and fail. It's not simply technical implementation, it's not only pure rationality: it's the implementation of ideas and of ideology. It's the slogans, narratives and metaphors that are able to quickly introduce ideas of practicality, elegance, completeness, symmetry and universality. It is a battle between the worlds of developers and scientists, and then winning over the interests of investors (both corporate and governmental) which will make a certain technology into an evolutionary stable concept. In the world of software developers it's about the transfer of knowledge, the informal meetings, the methodologies, the underground culture, and a sense of humor. In the world of capital, it is about giving back power and control, and creating viable business models. In the world of science, it's about the universe. Above all, making the machine work is a very complex game – but it's worth it.

Programming Language

Gardner

Building Code v0.0

Architecture schools around the globe are jumping on the parametric or computational design bandwagon, each having their departments experimenting with digital fabrication, generative and evolutionary algorithms. They fetishize flexible structures designed to move or change, and are enchanted by the aesthetics of smooth-surfaced, amoeba-shaped buildings. Of course, these are experimental projects that aim to explore new technologies – both their potentials and their limitations. But this engagement can be very akin to rabbits staring into the headlights of the car of technological progress rushing towards them, rendering them unable to move, completely struck by the spectacle. Beyond the fetish and the fad there is something missing: an involvement with the amalgam of space, concept, system, materiality, the social and the political at the heart of the architectural discipline. Architecture is a practice of unraveling socio-physical Gordian knots.

The rapid increase of network technology in the past decade has made these knots all the more interesting, as it has provided radical new perspectives and handles on their unraveling and re-assembling. What should not be underestimated, however, is that the methods for dealing with this changing playing field are shifting from form-making to code-making, from designing to programming, from a focus on aesthetics to a focus on performance.

The most powerful and influential designs are those which have become invisible. They have embedded themselves into our daily lives and are so omnipresent that we use them unconsciously. Building Code is one of these designs – or rather, systems – that has developed over time, a system to which many authors contributed. As an experiment let's entertain the thought for a moment that what we know as 'Building Code' is not the set of constraints and safety regulations with which architects and engineers comply on a daily basis in order to produce the built environment. Instead let's assume that Building Code is a programming language that generates built environment – generates form, interfaces between various systems, negotiates between social interests, and allows for hacking, tweaking and designing.

Imagine the building industry not as a conservative domain resilient to change, but rather more akin to the software industry, buzzing with new ideas, start-ups and collaborations. Perhaps it is difficult – and especially for those within it – to imagine the building industry as such. It is not an incomprehensible idea, but we would have to revisit the very practices which produce the built environment.

It is already evident that the computer boys have an interest in the built environment. They also have the best Trojan horse imaginable to penetrate any discipline they wish: it's the computer sitting on your – and everyone's – desk.

It is inevitable that the computer and the Building code, the *de facto* algorithm of our built environment, will eventually meet. So for now let's speculate upon the Building Code as a powerful programming language for the built environment.

Building Code would deal with machine-readable, networked laws and regulations. It would run on top of urban operating systems. It would be informed by civic information systems. *SimCity* would be seen as the communal touchstone for this image of the future – the future that those who would dream of Building Code will dream about. It is the dream of a city growing out of an algorithm, a city that generates processes that need to be managed, steered, guided, and controlled. But Building Code would be no mere simulation. Where *SimCity* is inhabited by Sims and viewed from a distance, Building Code would have to be much more than just the omni-perspective, it would also need to facilitate the citizens' perspective. It would not primarily be about making an interface between man and machine; rather, its real challenge would be to interface between the two dominant kinds of organization of human culture: the regulative – top-down – and the generative – bottom-up.

Shearing Layers WP2011

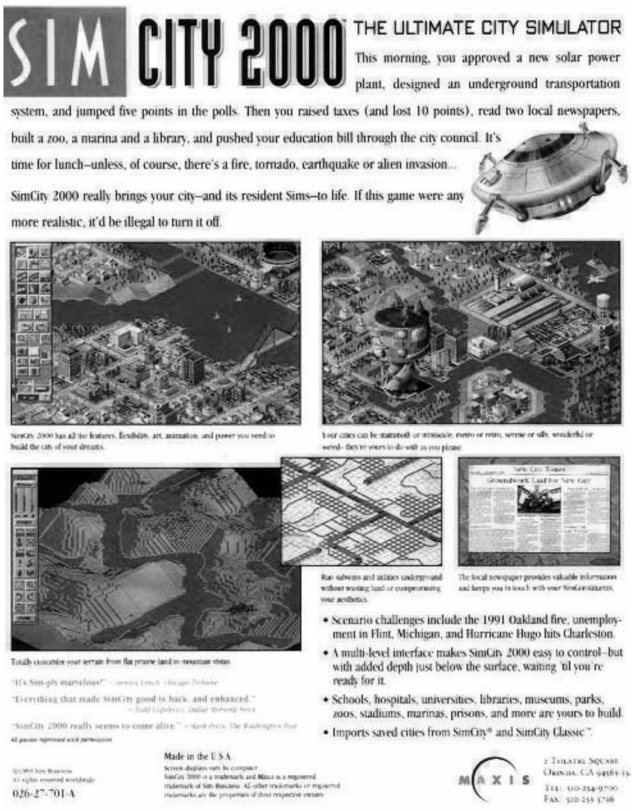
Shearing Layers is a concept that views buildings as a set of components that evolve in different timescales. Frank Duffy summarized this view: *"Our basic argument is that there isn't any such thing as a building. A building properly conceived is several layers of longevity of built components"*.

The concept is based on work in ecology and systems theory. The idea is that there are processes in nature which operate on different timescales and as a result there is little or no exchange of energy/mass/information between them. Stewart Brand transferred this intuition to buildings and noticed that traditional buildings were able to adapt because they allowed 'slippage' of layers; i.e., faster layers (services) were not obstructed by slower ones (structure).

Shearing Layers leads to an architectural design principle known as Pace-Layering which arranges the layers to allow for maximum adaptability.

SimCity WP2011

The game SimCity (1989) was originally developed by game designer Will Wright. The inspiration for SimCity came from a feature of the game Raid on Bungeling Bay that allowed Wright to create his own maps during development. Wright soon found he enjoyed creating maps more than playing the actual game, and SimCity was born. While developing SimCity, Wright cultivated a real love of the intricacies and theories of urban planning. He acknowledges the influence of System Dynamics developed by Jay Forrester whose book Urban Dynamics laid the foundations for the simulation. In addition, Wright also was inspired by reading The Seventh Sally, a short story by Stanisław Lem, in which an engineer encounters a deposed tyrant and creates a miniature city with artificial citizens for the tyrant to oppress.


SIM CITY 2000 THE ULTIMATE CITY SIMULATOR

This morning, you approved a new solar power plant, designed an underground transportation system, and jumped five points in the polls. Then you raised taxes (and lost 10 points), read two local newspapers, built a zoo, a marina and a library, and pushed your education bill through the city council. It's time for lunch–unless, of course, there's a fire, tornado, earthquake or alien invasion...

SimCity 2000 really brings your city–and its resident Sims–to life. If this game were any more realistic, it'd be illegal to turn it off.

SimCity 2000 has all the features, flexibility, art, animation, and power you need to build the city of your dreams.

Your cities can be mammoth or minuscule, metro or retro, serene or silly, wonderful or weird– they're yours to do with as you please.

Totally customize your terrain from flat prairie land to mountain vistas.

Run subways and utilities underground without wasting land or compromising your aesthetics.

The local newspaper provides valuable information and keeps you in touch with your SimConstituents.

- Scenario challenges include the 1991 Oakland fire, unemployment in Flint, Michigan, and Hurricane Hugo hits Charleston.
- A multi-level interface makes SimCity 2000 easy to control–but with added depth just below the surface, waiting 'til you're ready for it.
- Schools, hospitals, universities, libraries, museums, parks, zoos, stadiums, marinas, prisons, and more are yours to build.
- Imports saved cities from SimCity® and SimCity Classic™.

"It's Sim-ply marvelous!" – [illegible], Chicago Tribune

"Everything that made SimCity good is back, and enhanced." – Todd Copilevitz, Dallas Morning News

"SimCity 2000 really seems to come alive." – Mark Potts, The Washington Post

All quotes reprinted with permission

©1993 Sim Business
All rights reserved worldwide
026-27-701-A

Made in the U.S.A.
Screen displays vary by computer.
SimCity 2000 is a trademark and Maxis is a registered trademark of Sim Business. All other trademarks or registered trademarks are the properties of their respective owners.

MAXIS
2 Theatre Square
Orinda, CA 94563-[illegible]
Tel: 510-254-9700
Fax: 510-253-3736

Back cover of SimCity 200 box

Sims, The WP2011, WW2011

The Sims is a strategic life-simulation computer game first released in 2000. The inner structure of the game is actually an agent-based artificial life program. The presentation of the game's artificial intelligence is advanced and the Sims respond to outside conditions by themselves, although often the player/controller's intervention is necessary to keep them on the right track. The Sims technically has unlimited replay value in that there is no way to win the game and the player can play on indefinitely. It has been described as more like a toy than a game. A neighborhood in The Sims consists of a single screen displaying all playable houses.

In addition, the game includes a very advanced architecture system. The game was originally designed as an architectural simulation alone inspired by Christopher Alexander's Pattern Language, with the Sims there only to evaluate the houses, but during development it was decided that the Sims were more interesting than originally anticipated and their initially limited role in the game was developed further.

"The Sims really started out as an architectural game – you were designing a house and then the people were the scoring system. They came in and you were looking at how happy they were and how efficiently your house met their needs."

Third Culture NC2007

"Even a 'three cultures' view of human knowledge and ability is a simple model. However, contrasting design with the sciences and the humanities is a useful, if crude, way of beginning to be more articulate about it. Education in any of these 'cultures' entails the following three aspects:

- *the transmission of knowledge about a phenomenon of study*
- *a training in the appropriate methods of enquiry*
- *an initiation into the belief systems and values of the culture*

If we contrast the sciences, the humanities, and design under each aspect, we may become clearer of what we mean by design, and what is particular to it.

The phenomenon of study in each culture is
- *in the sciences: the natural world*
- *in the humanities: human experience*
- *in design: the artificial world*

The appropriate methods in each culture are
- *in the sciences: controlled experiment, classification, analysis*
- *in the humanities: analog, metaphor, evaluation*
- *in design: modelling, pattern-formation, synthesis*

The values of each culture are
- *in the sciences: objectivity, rationality, neutrality, and a concern for 'truth'*
- *in the humanities: subjectivity, imagination, commitment, and a concern for justice'*
- *in design: practicality, ingenuity, empathy, and a concern for 'appropriateness'*

In most cases, it is easier to contrast the sciences and the humanities (e.g., objectivity versus subjectivity, experiment versus analogy) than it is to identify the relevant comparable concepts in design. This is perhaps an indication of the paucity of our language and concepts in the 'third culture', rather than any acknowledgement that it does not really exist in its own right. But we are certainly faced with the problem of being more articulate about what it means to be 'designerly' rather than to be 'scientific' or 'artistic'.

Perhaps it would be better to regard the 'third culture' as technology rather than design. This 'material culture' of design is, after all. the culture of the technologist – of the designer, doer and maker. Technology involves a synthesis of knowledge and skills from both the sciences and the humanities, in the pursuit of practical tasks; it is not simply 'applied science', but 'the application of scientific and other organised knowledge to practical tasks…' (Cross. et al, 1981). The 'third culture' has traditionally been identified with technology. For example, the philosopher A.N. Whitehead (1932) suggested that: 'There are three main roads along which we can proceed with good hope of advancing towards the best balance of intellect and character: these are the way of literary culture, the way of scientific culture, the way of technical culture. No one of these methods can be exclusively followed without grave loss of intellectual activity and of character.' "

Two Cultures CPS1990

"Two polar groups: at one pole we have the literary intellectuals, who incidentally while no one was looking took to referring to themselves as 'intellectuals' as though there were no others. I remember G. H. Hardy once remarking to me in mild puzzlement, some time in the 1930s: "Have you noticed how the word 'intellectual' is used nowadays? There seems to be a new definition which certainly doesn't include Rutherford or Eddington or Dirac or Adrian or me. It does seem rather odd, don't y'know?". Literary intellectuals at one pole-at the other scientists, and as the most representative, the physical scientists. Between the two a gulf of mutual incomprehension – sometimes (particularly among the young) hostility and dislike, but most of all lack of understanding. They have a curious distorted image of each other. Their attitudes are so different that, even on the level of emotion, they can't find much common ground.

(…) The non-scientists have a rooted impression that the scientists are shallowly optimistic, unaware of man's condition. On the other hand, the scientists believe that the literary intellectuals are totally lacking in foresight, peculiarly unconcerned with their brother men, in a deep sense anti-intellectual, anxious to restrict both art and thought to the existential moment. And so on. Anyone with a mild talent for invective could produce plenty of this kind of subterranean back-chat. On each side there is some of it which is not entirely baseless. It is all destructive. Much of it rests on misinterpretations which are dangerous." – C .P. Snow

This quote is from the 1959 Rede Lecture by British scientist and novelist C. P. Snow. Its thesis was that the breakdown of communication between the 'two cultures' of modern society – the sciences and the humanities – was a major hindrance to solving the world's problems. As a trained scientist who was also a successful novelist, Snow was well placed to articulate this thesis.

Unconference WP2011

The term unconference, indicating a participant-driven meeting, has been applied, or self-applied, to a participant-driven meeting. The term unconference has been applied, or self-applied, to a wide range of gatherings that try to avoid one or more aspects of a conventional conference, such as high fees, sponsored presentations, and top-down organization. BarCamp, Foo Camp, Mashup Camp and Bloggercon are examples of unconferences.

Universal Machine MM2011

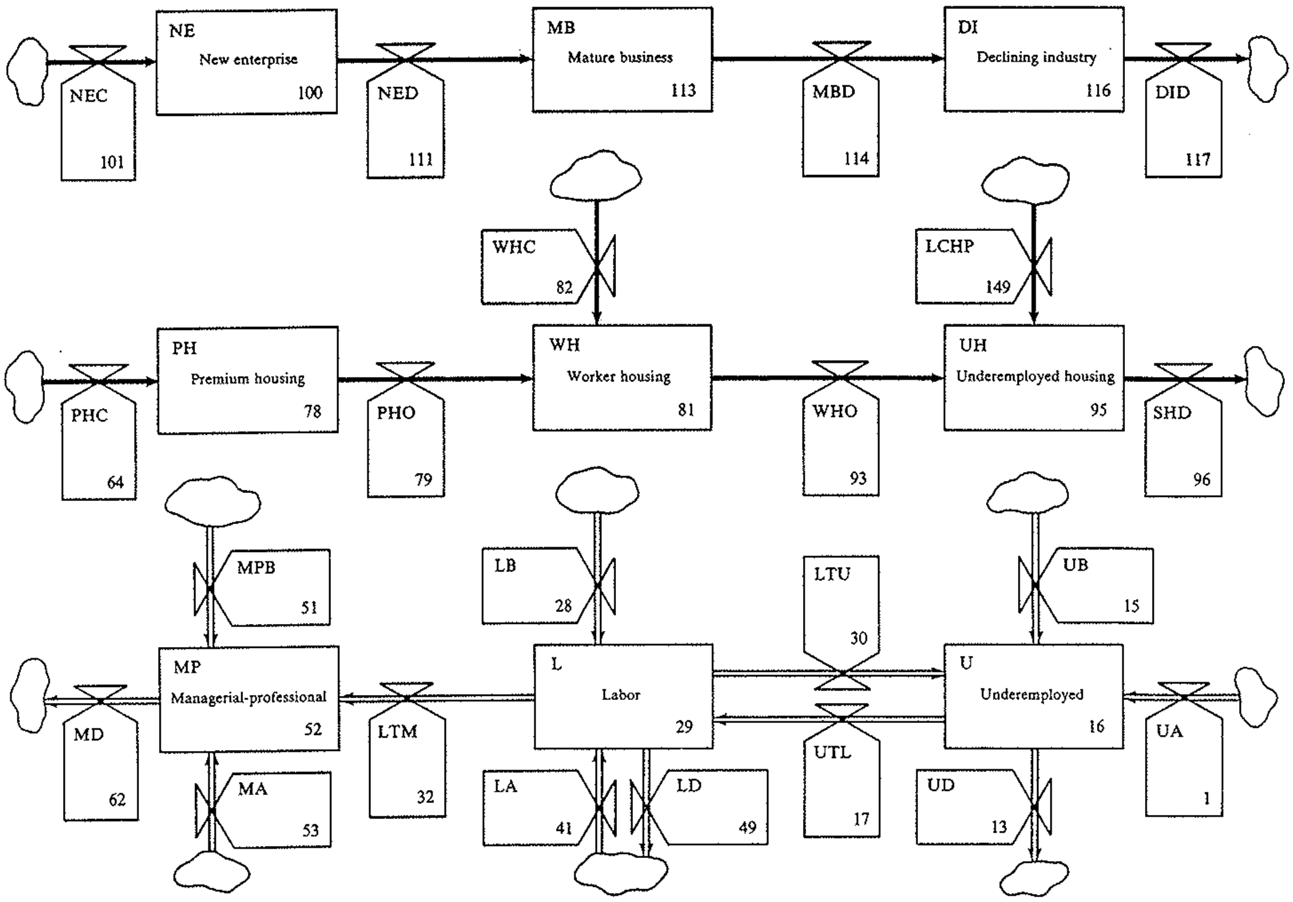
It must be understood that the idea of the universal machine – or, more precisely, the idea of computation itself – is not a mechanical device, not a computer and not a machine. It is a (mathematical) model of a computation; it is Alan Turing's thought experiment given to all of us. It proposes a theoretical device that manipulates symbols on an infinite strip of tape according to a table of rules. It tells us that a very simple device can solve any 'reasonable' problem in a finite period of time (but perhaps not in our lifetime), and it allows us to rationalize nature in computational models in addition to the mathematical models that are currently dominant in science.

The idea of a computable universe can be traced from Leibniz to Konrad Zuse to Edward Fredkin and to Stephen Wolfram, who claims that the universe may be a cellular automata (another equivalent model of computation together with: lambda calculus, combinatory logic, Markov algorithm, register machine, etc.).

Urban Dynamics WP2011, JF1969

Jay Forrester's book *Urban Dynamics* (1969) models a city as a set of nodes connected through feedback loops.

The nodes would be entities like 'industry', 'worker housing', 'premium housing', 'labor', 'underemployed', 'managerial profession'. Urban Dynamics strongly influenced game-designer Will Wright when he developed the city simulation game SimCity which was released in 1989.



A diagram from Urban Dynamics (JF1969) depicting a part of Forrester's Urban Model. The major levels (rectangles) and rates (valve symbols) for the model of an urban area

Wright, Will WP2011

Will Wright (1960) is an American video game designer and co-founder of the game development company Maxis, now part of Electronic Arts. In April 2009 he left Electronic Arts to run 'Stupid Fun Club', an entertainment think tank in which Wright and EA are principal shareholders.

The first computer game Wright designed was Raid on Bungeling Bay in 1984 but it was SimCity that brought him to prominence. The game was released by Maxis, a company Wright formed with Jeff Braun, and he built upon the game's theme of computer simulation with numerous other titles including SimEarth and SimAnt.

Wright's greatest success to date came as the original designer for the Sims games series which, as of 2009, was the best-selling PC game in history. The game spawned multiple sequels and expansions and Wright earned many awards for his work. His latest work, Spore, was released in September 2008 and features play based upon the model of evolution and scientific advancement.

Zuse, Konrad JS2011

Konrad Suze (1910-1995) not only built the first programmable computers (1935-1941) and devised the first higher-level programming language (1945), but also was the first to suggest (in 1967) that the entire universe is being computed on a computer, possibly a cellular automaton. He referred to this as 'Rechnender Raum' or Computing Space or Computing Cosmos. Many years later similar ideas like digital physics were also published, popularized and extended by Edward Fredkin (1980s), Jürgen Schmidhuber (1990s), and more recently Stephen Wolfram (2002).

Sources

BLP2007 Barry B. LePatner, *Broken Building, Busted Budgets – How to Fix America's Trillion-Dollar Construction Industry* (The University of Chicago Press, 2007)

CA1964 Christopher Alexander, *Notes on The Synthesis of Form* (Harvard University Press, 1964)

CA1977 Christopher Alexander, Sara Ishikawa, Murray Silverstein, *A Pattern Language* (Oxford University Press, 1977)

CA1996 Christopher Alexander, *The Origins of Pattern Theory the Future of the Theory, And The Generation of a Living World.* http://www.patternlanguage.com/archive/ieee/ieeetext.htm (accessed 22 may)

CE1999 Charles M. Eastman, *Building Product Models: Computer Environments Supporting Design and Construction.* (CRC Press, 1999)

CPS1990 C. P. Snow, *The Two Cultures*, Leonardo, Vol. 23, No. 2-3, New Foundations: Classroom Lessons in Art/ Science/ Technology for the 1990s (The MIT Press, 1990) pp. 169-173

DP1994 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1994)

DR2010 Douglas Rushkoff, *Program or be Programmed, Ten Commands for a Digital Age* (OR Books, 2010)

FG2010 Fabio Gironi, *Science-Laden Theory*, Speculations I (2010)

FT2006 Fred Turner, *From Counterculture to Cyberculture – Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism* (University of Chicago Press, 2006), page 106.

JF1969 Jay Forrester, *Urban Dynamics* (Pegasus Communications, 1969)

JB1989 James Beniger, *The Control Revolution: Technological and Economic Origins of the Information Society* (Harvard University Press, 1989)

JBIM2008 *Journal of Building Information Modeling* (Spring 2008)

JL2011 John Law, *Actor Network Theory and Material-Semiotics*, 25 April 2007, heterogeneities.net (accessed May 22, 2011)

JS2011 *Zuse's Thesis: The Universe is a Computer*, http://www.idsia.ch/~juergen/digitalphysics.html (accessed June 6, 2011)

KBWC1987 *Using Pattern Languages for Object-Oriented Programs* – Submitted to the OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming.

MW2011 Wolfram Mathworld, http://mathworld.wolfram.com/CellularAutomaton.html (accessed June 1, 2011)

MM1969 Marshall McLuhan *The Playboy Interview: Marshall McLuhan*, Playboy Magazine (March 1969)

MCK2003 Martin Campbell-Kelly, *From airline reservations to Sonic the Hedgehog: a history of the software industry* (MIT Press, 2003)

NE2010 Nathan Ensmenger, *The computer boys take over: computers, programmers, and the politics of technical expertise* (MIT Press, 2010)

NC2007 Nigel Cross, *Designerly Ways of Knowing* (Birkhäuser, 2007)

OM2011 – Om Malik, *What Do Don Draper and GitHub Have In Common?*, http://gigaom.com/2011/04/07/what-do-don-draper-and-github-have-in-common/ (accessed May 29, 2011)

RB1990 Reyner Banham, *A Black Box, the secret profession of architecture*; (New Statesman & Society 12 October 1990)

SG1983 Stephen Grabow, *Christopher Alexander, The Search for a New Paradigm in Architecture* (Oriel Press, 1983)

SL1984 Steven Levy, *Hackers: Heroes of the Computer Revolution* (Doubleday, 1984)

SF2006 Stephen Fer, *Event-Driven Programming: Introduction, Tutorial, History* (2006)

WW2011 Will Wright interviewed by William Wiles for IconEye http://www.iconeye.com/index.php?option=com_content&view=article&id=3068:will-wright-interview (Accessed May 30, 2011)

WP2011 Wikipedia, http://wikipedia.org (accessed May & June, 2011)

WWW2011 WikiWikiWeb, http://c2.com/cgi/wiki (accessed May & June 2011)

Acknowledgement

A special thanks to Steven Chodoriwsky and Dubravka Sekulić

Design by Oscar David Quijano Robayo

Tracing Concepts is produced by Marcell Mars and Edwin Gardner and supported by the Jan van Eyck Academie.

The programme of the Jan van Eyck Academie is funded by the Ministry of Education, Culture and Science, the province of Limburg and the City of Maastricht.

JVE

www.janvaneyck.nl