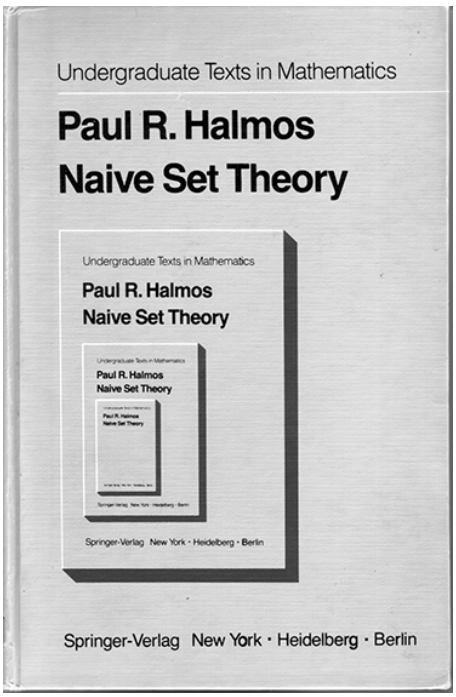
David Reinfurt, Infinite Versioning

German psychologist Max Wertheimer wrote in 1909:

> I break a stick in two. One approach says, I now have "two."
> (Two what? That's immaterial, I have two—new—units.)
> The arithmetic makes a jump: first there is one, a stick, then there are two... and between the first and the second state of affairs there is a gap, which is ignored.

But we'll call a stick a stick. Better yet, let's call it one stick then two, not forgetting that "one" and "two" are words and therefore merely point to external concepts. It doesn't help much using the digits "1" and "2" instead—they're symbols, too. They stand in for something else. That something is the concept of number and in the simplest case, it might be described as a measure of quantity, but in fact, it is a good deal weirder. We are used to how words shift; they're context-sensitive, subjective, essentially relative, and they "mean" by marking their proximity and distance to other words. But numbers are supposed to be definite, absolute, eternal. We grow up learning that numbers aren't like words at all. Wertheimer, for one, knew this assumption was wrong: numbers can be relative, as well, and parts don't always sum neatly to wholes. He was well-acquainted with the mathematical-philosophical inquiry into the essence of number published in 1893 by German mathematician Richard Dedekind. Dedekind intuited that numbers are not tied concretely to measurable magnitude. He lays out his thesis in his preface to "The Nature and Meaning of Numbers:"

> Numbers are the free creations of the human mind; they serve as a means of apprehending more easily and more sharply the *difference* of things.



He continues by describing number as a consequence of the "ability of the mind to relate things to things." Dedekind hinged his argument on Naïve Set Theory, which his friend and colleague Georg Cantor was developing around the same time. Set theory is the root-level logic of mathematics, sitting below basic arithmetic, number, even counting. Fundamentally, set theory concerns the study of groups of things, or sets and their members. For example, a set of fruit can be described with the members {apple, orange, banana, mango}, or {car, truck, bicycle, horse} constitutes a set of vehicles. To define number rigorously, Dedekind asked what property these two sets share. If you pair each fruit with a specific vehicle in a one-to-one relation, and this pairing exhausts both sets so that every member of the first goes with a member of the second, then what they share is their "number." In this case, it's 4.

Dedekind's invocation of number seems to be the wrong way around. He suggests (well, proves) that counting is not an innate reflex, and numbers don't identify immutable qualities. Numbers are not handed down from the astral plane, but are instead *produced* by the similarities and differences between sets. In this way, a number can be identified concisely as marking a *relation* between two things, or two sets of things.

Set theory abstracts these relations even further. It's a logical lasagna where sets may include other sets or even themselves. Two sets are considered equal if they have the same members. Although numbers arise from the correspondences of sets, it's possible and quite useful to have sets of numbers also. The infinite collection of positive whole integers (called the natural numbers) is one such set. Its members are listed as {1,2,3 ... ∞}. The set of irrational numbers, or those that cannot be described as a fraction of two whole numbers, is another. These two sets, the naturals and the irrationals, hint at how strange and stretchy the concept of number actually is. For example, the "number" of members in the set of naturals is ∞. Remove one member from the set, like "1," and the number of members remains ∞. As for the irrationals, none of its members can even be described precisely by existing digits. The best we can do is approximate their values using an infinite string of decimals or

assign a symbol to stand for that "number." π is a prime example. Other irrationals live in anonymity, without proper names of their own.

Two sets are called "similar" if one can be changed into the other continuously using the same function. It's easiest to understand by using sets of whole numbers. So, the sets {1,2,3,4} and {1,4,9,16} are said to be similar because every member of the first set can be transformed into every member of the second set by multiplying itself by itself. 1×1=1, 2×2=4 and so on.

Returning to the broken stick, a way out of the arithmetical conundrum appears. Instead of remaining fixed on counting discrete objects, we can think instead in terms of *transformations.* In place of one stick, a gap, and then two, we have the same material of one stick transformed into two different configurations. Instead of objects, we can think of moments or states.

Consider Microsoft Word's "track changes" command. This software function allows you to see immediately where, how, and when a text has been changed, and it attempts to make all the various moments in the life of the file visible. Or how about Apple's Time Machine? The system-level function makes a consistent backup of your computer's files at regular intervals. Entering the Time Machine interface allows you to magically roll back the clock and reset a specific file to a previously saved version, while leaving the rest of your computer in its present state. Wikipedia's recent changes can be easily compared to the current version from the sidebar menu. A software "diff" tool makes it crystal clear what has changed and what has not by comparing multiple versions and highlighting the differences. Web browsers retain your history of recently visited sites, and a page accessed today may be different from the same page tomorrow. There is even a meta-"track changes" for the whole Internet called the Wayback Machine. Type a URL into the search field at web.archive.org and the Wayback Machine recalls snapshots of the that website at regular intervals stretching back to 1996. Click on a particular date, and you're transported to that page, then. Hyperlinks work within the site and outside of it, allowing you to surf the 1996 web in 2013 on a 2012 MacBook Air running Safari v 6.0.3.

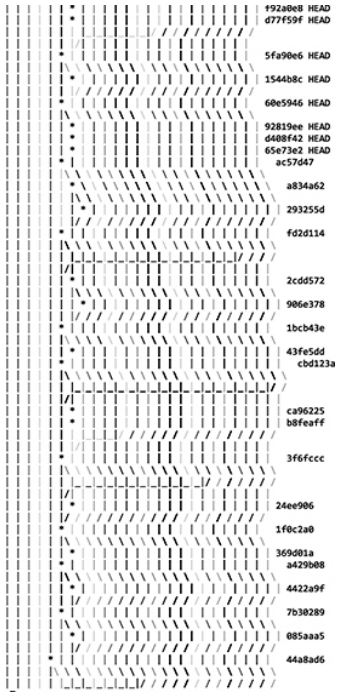
Tracking versions becomes considerably more complicated when making software instead of using it. Active software products are a work-in-progress, under constant revision. Each product release contains both additional features and new bugs. The bugs get fixed, a new version is released, and the process rolls on. Most software follows a versioning scheme that uses three numbers separated by periods, such as OS X 10.8.3. The first is used to identify a major change, the second a minor improvement, and the third to mark a slight revision. Other versioning schemes have come and gone, such as adding the year to the product name in Adobe Illustrator 88 or Microsoft Windows 95. Donald Knuth's TeX versions successive releases by adding one further digit in the infinite sequence of a decimal approximation of π. Knuth says the software, currently in v 3.1415926, dies with him. The project will be frozen, and the version number changed from the current string of approximate digits to its purely irrational symbol.

Versioning becomes hairier yet when writing code. A commercial software project involves a potentially large team that is not necessarily working in the same room or even at the same time. The development process is logistically baroque, and also modular. Modern programming languages are built on an object-oriented model in which chunks of code are treated as black boxes that can be developed individually, plugged into a larger framework, revised

separately, and repurposed. Software engineering lacks the linear clarity of an assembly line—its workflows are knotty and opaque. Small changes in one part of the code can have large effects elsewhere in the project, so keeping track of who wrote what, when is mission critical. Version Control Systems (VCS) are fit to the task, and allow a large group to work on shared files simultaneously without stepping on each other's toes or rewriting their functions. An early version control system named Source Code Control System (SCCS) appeared in 1972 at Bell Labs. It was updated and replaced by an alphabet soup of protocols including RCS (1982), CVS (1986), CVSNT (1998), and SVN (2000). The fact that these were so quickly outdated themselves highlights the thorniness of the problem as software projects continued to grow larger and more distributed. By the time Finnish programmer Linus Torvalds issued this veiled invitation in 1991:

> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu)...

To launch what would become the Linux operating system, projects had grown beyond all reasonable capacity to corral the complexity of their development process. Linux is an open source project built from hundreds of thousands of contributions by volunteer coders coordinated through an online codebase. The current release is v 3.8.8. It contains more than 12 million lines of code spread across 36,000 discrete files. As the project scaled, Torvalds became frustrated with the existing version control softwares, so he wrote a new one from scratch in 2005 called Git.



Git has been quickly and widely adopted, in part, because it's an explicitly *distributed* version control system. Rather than one central repository (typically stored on a server) that holds the master project files, Git works with any number of linked repositories. Each instance of the code, each repository, is equivalent, and there is no "master." Instead, Git identifies the master as a complex set of versioned relations among the individual copies. The software

brokers the required transactions to keep each copy up-to-date using commands like push, pull, clone, branch, merge, and commit. Repositories can also be stored online at sites like Github.com, a repository of repositories, which become massive, publicly-shared code warehouses. Unlike previous version control systems, Git doesn't track the history of any particular file, but instead monitors all of the raw project data (regardless of which files it is written into), making snapshots of what the whole looks like at any specific time. This is perhaps the most radically powerful part of Git's model, as it doesn't track changes, it monitors transformations. Scrolling back to a previous version is simple—comparing snapshots is done via the diff command, and a graph, drawn as a tree with branches, commits and merges, which makes it possible to see what has changed and what has stayed the same.

Swiss product designer Max Bill addressed versioning 50 years before in his essay "Continuity and Change." Bill was writing about hardware not software, but the terms are familiar. He marked out continuity as that which persists and change as that which mutates. He limited the domain of these terms to a particular product category:

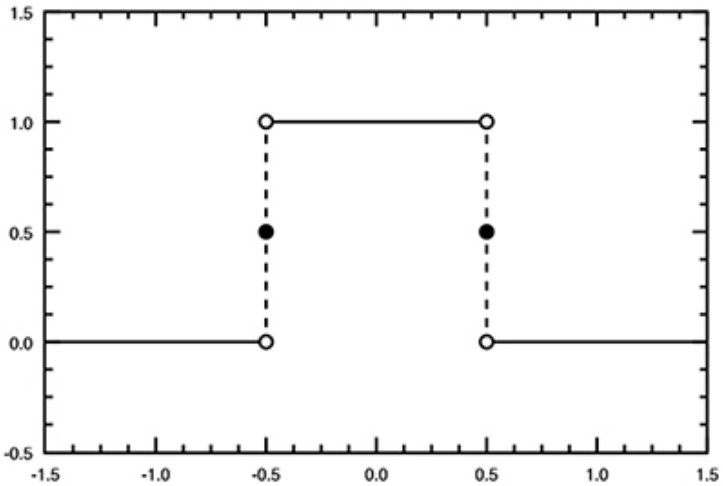
> We judge a chair, for example, not only in terms of its individual qualities but also as a representation of its type—chairs as a whole over a longer time-span.

For Bill, form embodied the harmonious, stable arrangement of a product's relations. These included not only the specific form choices made in designing, say, a watch, but also the relation between a specific watch and the entire category of "watches." Good design yields the most "typical," valid form, completely in accord with its type. However, he acknowledged that it was not nearly so simple, and pointed to Viennese architect Adolf Loos's thoughts on top hats from "Regarding Economy" from 1924:

> A top hat can take a variety of forms. Imagine a row of a hundred of them. I want to go to a funeral. I try various shapes and see that most of them are impossible, ridiculous, and that only one hat fits. The 1924 hat, let's say. This hat is the only possible one for me and the time in which I live.

Loos writes off this counter-example to fashion and proceeds in his appeal, saying, "I reject any form of innovation-mania." In the next paragraph, he continues,"We should measure beauty in terms of time." Loos advocated an incremental and eternal approach for design, but still he recognized that a form, like the 1924 top hat, might be valid only at one particular moment. He didn't, however, allow for a less linear accounting. Time is not nearly so straight. (Just look at the overlapping versions and tangled branches of a Git project graph.) Any form and its judgment at a specific moment is valid for that time only and its standing may be changed retroactively in the future: A new form appears, affecting the whole class and each member's standing within it, effectively re-centering the category. Bill suggested morphology, the study of form over time, would be the best tool to understand this versioning process.

Morphological studies can be carried out empirically and visually, but there is an even more definite flavor: mathematical morphology is a formalized language for identifying transformations over time. Its mother-tongue is set theory. Mathematical morphology studies whole groups and their shifting relations to understand how any one form changes in time. A typical application is digital image processing using a method called the Top Hat Transform. The shape of the function graphs a line that resembles its name.



Applied to the two-dimensional grid of a greyscale image, the function compares its pixel values to a bounding structure, a "master" of sorts.
The Top Hat Transform draws out areas of similarity and areas of difference and produces one composite image that shows what has changed and what has not. It may be applied across an unlimited set of images at once, revealing how their forms relate. You can think of it as Git's diff command set to work on images.

Version control is useful for hardware (products) and essential for software. But what if the thing that is changing is neither one nor the other? With the recent appearance of 3-D printing, "real" objects can be made directly from digital data. The technology is becoming cheaper and more widespread, resulting in an expanding community of things that are neither hard nor soft, exactly. These are mongrels, whose forms are constituted of the relations between the data that define them and when and how they were produced. Make one small adjustment to the computer model, hit "print," and you have a new form. As the source material is digital, to morph between any two particular instances is computationally direct, and also continuous.

It's no giant leap to imagine that the two-dimensional Top Hat Transform might be expanded to address morphological changes in 3-D-printed objects over time. We might be able to visualize instantly the entire geometric history of a certain form. A specific path through time can be programmatically collected, assembled into a model, and printed to produce, say, a fourth-dimensional teacup with all its forms and times embedded. Alternately, we can imagine a time-lapse model that would trace the teacup's evolution.

Now, the text you are reading right now originally formed the third part of a longer essay called "G-E-S-T-A-L-T," published online as a PDF by The Serving Library and released June 10, 2013 at 7:35 PM[1]. The essay was then printed and published in BULLETINS OF THE SERVING LIBRARY, #5.
Two separate print runs of BULLETINS were produced and distributed in the United States by The Sheridan Group and in Europe by Sternberg Press. In the meantime, "Infinite Versioning" was re-channeled as the narration for a slide lecture of the same name delivered in Venice, Italy on May 30, 2013. The lecture happened before the text was released online, although by this point it was already complete. This slide-show text was then slightly adjusted and delivered at The Banff Centre in Alberta, Canada on July 23, 2013. Approximately one month earlier, "Infinite Versioning" was reworked and used to accompany a set of photograms made by Jason Fulford for a forthcoming book to be published by Blind Spot (New York) in early 2014 and titled, "Gestalt, or The Whole Enchilada." In the midst of all this, a number of things have changed in this version, including its ending.

As it turns out, Git (or rather, its online community hosted at www.github.com) has developed a software tool that closely matches what I described two paragraphs previous. They call it "3D File Diffs"[2] and it offers users working together on a single product design project hosted through the website, a way to see what has physically changed in the design, *over time.* The viewer compares transformations in a 3-dimensional data set, and renders the information onscreen through an interactive interface. The Diff Viewer allows users to manipulate the object in 3d space, color coding changes made in susbsequent versions. You can even scrub back and forth through the design over time using the aptly-named "Revision Slider."

But it's unlikely that we really need such technological hijinks. We might simply be better off developing slower eyes—a new vision that would allow us to

see whole shapes in longer exposures assembled over time. This sensation of pure, object-less movement, a transformation in itself, marks the demilitarized zone between things and processes. Here, objects are usefully suspended, appearing as integrally related sets of transformations rather than any single, resolved composition. It's not as difficult as it might sound. We've already trained ourselves to see the motion of a swinging door, the path of a boomerang, the flight of an airplane, or the trajectory of a thought as it shifts and mutates through any number of discrete versions.

01. "G-E-S-T-A-L-T" is published at http://www.servinglibrary.org/read.html?id=77066
02. 3D File Diffs and the viewer are described and demonstrated on GitHub's blog at https://github.com/blog/1633-3d-file-diffs

Published on <o> future <o>, June 9, 2014.