Note

Regulation by Software

James Grimmelmann

CONTENTS

In real space we recognize how laws regulate—through constitutions, statutes, and other legal codes. In cyberspace we must understand how code regulates—how the software and hardware that make cyberspace what it is *regulate* cyberspace as it is. As William Mitchell puts it, this code is cyberspace's "law." *Code is law*.[1]

And finally the architecture of cyberspace, or its *code*, regulates behavior in cyberspace. The code, or the software and hardware that make cyberspace the way it is, constitutes a set of constraints on how one can behave.[2]

INTRODUCTION

Six years ago, Lawrence Lessig had two insights. First, code regulates. Computer software ("code") can constrain behavior as effectively as law can. Second, code is like physical architecture. When software regulates behavior online, it does so in a manner similar to the way that physical architecture regulates behavior in the real world.[3] His catchphrase—"code is law"—is shorthand for the subtler idea that code does the work of law, but does it in an architectural way. With this one phrase and two ideas, he opened up an entire line of study: how regulating through software rather than through law changes the character of regulation.

Unfortunately, that line of study has been stunted, and in a sense, it is Lessig's fault—for having three insights, instead of stopping with two. In the book that made "code is law" famous, *Code and Other Laws of Cyberspace*, Lessig also argued that software itself can be effectively regulated by major social institutions, such as businesses or governments. He then completed the syllogism. If other institutions can regulate software, and software can regulate individual behavior, then software provides these institutions an effective way to shape the conduct of individuals.

This third insight gives the first two their urgency, but its salience has diverted attention from them. *Code* argues that decisions about the technical future of the Internet are important questions of social policy, because these decisions will have the force of law even as they defy many of our assumptions about law.[4] This argument has attracted great attention. Many

1. LAWRENCE LESSIG, CODE AND OTHER LAWS OF CYBERSPACE 6 (1999) (endnote omitted).
2. Lawrence Lessig, *The Law of the Horse: What Cyberlaw Might Teach*, 113 HARV. L. REV. 501, 509 (1999).
3. The most important intellectual precursor of Lessig's popular formulation was Joel R. Reidenberg, Lex Informatica*: The Formulation of Information Policy Rules Through Technology*, 76 TEX. L. REV. 553 (1998).
4. LESSIG, *supra* note 1, at 58-60.

scholars have debated Lessig's claim that sufficient outside pressure could modify the Internet's basic technological structure.[5] Others, taking this claim as given, have debated what changes our governing institutions should make to the Internet's systems.[6]

In the face of such questions, the minor premise of *Code*—that software is a form of regulation akin to physical architecture—has gone comparatively unexamined. Scholars make a few cursory nods in Lessig's direction, note the power of architectural constraints, and then launch immediately into their arguments about one particular problem or another. This Note begins the process of correcting that oversight.

Lessig's insight that code resembles physical architecture is incomplete. Just as code is like law but not really the same, code is like physical architecture but not really the same. The structures that can be built out of software are so much more intricate and fragile than the structures that can be built out of concrete that there is a qualitative difference between them. You can hack a computer program into destroying itself; you will have no such luck hacking a highway. Computer software is its own distinctive modality of regulation, and it needs to be treated as such.

In particular, a systematic study of software as a regulator reveals several recurring patterns. These patterns are present in any regulation by software, whether it involves privacy, intellectual property, free speech, online jurisdiction, or commerce, to name just a few fields profoundly affected by the growing use of software. We can evaluate the success or failure of regulation by software in one of these areas by asking whether these patterns run with the grain of our regulatory goals or against them. For designers, the patterns provide a road map for the appropriate construction of software systems that must do "legal" work. For analysts, they provide a toolkit of ways to predict the consequences of turning a software system loose on real people.

Part I situates software within Lessig's theory of different and complementary modalities of regulation that constrain individuals. In *Code*,

5. *See, e.g.*, Michael Geist, *Cyberlaw 2.0*, 44 B.C. L. REV. 323 (2003); Charles Fried, *Perfect Freedom or Perfect Control?*, 114 HARV. L. REV. 606 (2000) (book review). Lessig himself was responding to a line of arguments that began with David R. Johnson & David Post, *Law and Borders—The Rise of Law in Cyberspace*, 48 STAN. L. REV. 1367 (1996). A notable early reply was James Boyle, *Foucault in Cyberspace: Surveillance, Sovereignty, and Hardwired Censors*, 66 U. CIN. L. REV. 177 (1997). More recent discussions of whether cyberspace can be effectively regulated through technology mandates include Peter Biddle et al., *The Darknet and the Future of Content Protection*, *in* DIGITAL RIGHTS MANAGEMENT: ACM CCS-9 WORKSHOP, DRM 2002, at 155, 174-75 (Joan Feigenbaum ed., 2003), *available at* http://crypto.stanford.edu/DRM2002/darknet5.doc; and Susan P. Crawford, *The Biology of the Broadcast Flag*, 25 HASTINGS COMM. & ENT. L.J. 603, 643-51 (2003).

6. *See, e.g.*, Yochai Benkler, *Net Regulation: Taking Stock and Looking Forward*, 71 U. COLO. L. REV. 1203 (2000); Edward Lee, *Rules and Standards for Cyberspace*, 77 NOTRE DAME L. REV. 1275 (2002); Lawrence B. Solum & Minn Chung, *The Layers Principle: Internet Architecture and the Law*, 79 NOTRE DAME L. REV. 815 (2004).

he postulates four such modalities: law, social norms, markets, and physical architecture. He then argues that software is a subspecies of physical architecture as a modality. I argue instead that three basic characteristics of software establish it as a distinct modality that should not be conflated with any of the others:

Software is *automated*. Once set in motion by a programmer, a computer program makes its determinations mechanically, without further human intervention.

Software is *immediate*. Rather than relying on sanctions imposed after the fact to enforce its rules, it simply prevents the forbidden behavior from occurring.

Software is *plastic*. Programmers can implement almost any system they can imagine and describe precisely.

Software is like physical architecture and unlike law in being automated and immediate. However, plasticity is more characteristic of legal systems than of architectural ones. Software's plasticity interacts with its automation and its immediacy to produce consequences that set it apart from both law and physical architecture.

In Part II, I turn to these distinctive consequences. There are four recurring and predictable patterns present in any regulation by software:

First, along the traditional continuum between rules and standards, software lies at the extreme rule-bound end. Because a computer, rather than a person, makes a program's decisions, rules encoded in software are free from ambiguity, discretion, and subversion.[7] The plasticity of software means that even intricate combinations of highly particular rules remain formally realizable.

Second, software can regulate without transparency. Frequently, those regulated by software may have no reasonable way to determine the overall shape of the line between prohibited and permitted behavior. The plasticity of software and its automated operation also bedevil attempts to have software explain itself. Even experts may not understand why a program acts as it does.

Third, software rules cannot be ignored. Parties facing a decision made by software can, at best, take steps to undo what software has wrought. In contrast, legal default rules can often be ignored because they operate only when one party invokes the legal system. This difference means that regulation by software creates different transaction costs than regulation by law does.

Fourth, software is more fragile than other systems of regulation. Hackers can turn its plasticity against it, and its automated operation means

7. Not every rule can be implemented by software. Software is so rule bound that from its perspective some "rules" seem more like standards.

that unintended consequences are shielded from human review. Its immediacy also speeds up failures. A program can go from perfectly functional to completely broken with the flip of a single bit. These effects combine to make software more prone to sudden, unexpected, and severe failures than other regulators.

By running down the list of patterns, we can quickly predict the consequences of using software in a particular way to solve a particular problem. We can also evaluate these consequences normatively, to say that software is a good or a poor choice for solving a given problem. Sometimes extreme ruleishness is a desirable feature of a regulatory system. Sometimes it is not. Sometimes if a system fails, human life is at risk. Sometimes nothing more than mild inconvenience will result. Thus, these four patterns provide a general methodology for assessing the use of software in a given regulatory context.

Part III explains this methodology and applies it to two case studies. The methodology predicts that software is a good way to manage negotiations and transactions in online marketplaces such as online auction sites and electronic stock exchanges. The underlying rules of these marketplaces are typically simple and well understood. Software rules can track these market rules well, as predicted by the first pattern. On the other hand, the methodology predicts several pitfalls for the use of software to restrict the distribution of digital media. Here, the idea has been to use "digital rights management" (DRM) software to enable some kinds of consumption of media while simultaneously preventing some kinds of redistribution. Both the transaction cost implications of the third pattern and the failure mode implications of the fourth are troubling here.

I. SOFTWARE AS A MODALITY OF REGULATION

This Part describes Lessig's framework of modalities of regulation and then situates software within that framework, first according to Lessig's (partially correct) view of software as a form of physical architecture and then as its own independent modality.[8]

A. *Modalities of Regulation*

People's behavior is subject to many different kinds of constraints, of which law is only one. Under different circumstances, one or another kind of constraint may be the most significant in shaping the choices available to

8. The phrase "modalities of regulation" itself comes from Lessig's *The New Chicago School*, which worked out the concept's intellectual genealogy and placed on the table topics such as the nature of a regulatory modality and the relationship between different modalities. Lawrence Lessig, *The New Chicago School*, 27 J. LEGAL STUD. 661, 663 (1998).

people. These kinds thus represent different tools available to those who wish to push behavior in one direction or another. To understand what we gain in understanding by theorizing them as modalities of regulation, it is easiest to start by considering the four modalities Lessig discusses in *Code*: law, social norms, markets, and physical architecture.[9]

That *law* constrains behavior is obvious. Law threatens those who misbehave with punishment. It encourages individuals to choose the "right" course of action by associating sufficient sanctions with the "wrong" one to make it comparatively unattractive.[10]

Even in the absence of formal law, informal *social norms* also constrain behavior.[11] Human behavior in everyday life is extensively programmed by informal but strong codes of conduct: Don't maintain eye contact with strangers,[12] follow instructions from authority figures,[13] and so on. Stronger sanctions back up more restrictive rules. Lynching in the South, for example, was historically a constraint on interracial sex.[14]

A third form of constraint is a *market*. Unlike laws and social norms, which threaten or cajole, markets place an explicit advance price tag on particular behavior.[15] From Smith to Marx and beyond, economists have described how price constraints shape human behavior. Policymakers create markets in pollution permits and in radio station licenses to regulate the use of scarce resources more efficiently; truth-in-advertising laws and antitrust suits are legal interventions that aim to improve the regulation implicitly carried out by existing markets.

The last modality of regulation Lessig describes is *physical architecture*. That physical reality, particularly the human-built environment, is a constraint on behavior is beyond obvious. The "laws" of physics, unlike positive laws, cannot be broken. The work of architects and urban planners rests on the further assumption that such physical constraints can be deployed as a purposive tool of a regulatory agenda, that different

9. *See* LESSIG, *supra* note 1, at 235 app.

10. *See* OLIVER WENDELL HOLMES, THE COMMON LAW 40 (Mark DeWolfe Howe ed., Belknap Press 1963) (1881).

11. *See generally* ERIC A. POSNER, LAW AND SOCIAL NORMS (2000).

12. *See* ERVING GOFFMAN, BEHAVIOR IN PUBLIC PLACES: NOTES ON THE SOCIAL ORGANIZATION OF GATHERINGS 84 (1963).

13. *See* STANLEY MILGRAM, OBEDIENCE TO AUTHORITY: AN EXPERIMENTAL VIEW 1-12 (1974).

14. *See, e.g.*, JAMES GOODMAN, STORIES OF SCOTTSBORO (1994) (illustrating complex, racially charged norms of sexual conduct in the context of the famous "Scottsboro Boys" trials).

15. Of course, the market itself depends on laws and social norms to prevent people from using a resource without paying the price, and those resources may themselves be creatures of law. In such cases, the market is the predominant modality of regulation in the sense that the price tag is the most salient feature of the constraint set. People think in terms of how to pay the price rather than how to shoplift.

spaces conduce to different behaviors.[16] For example, Lessig describes speed bumps as an architectural constraint on speeding.[17]

Each of these three nonlegal modalities has generated a rich "law and *X*" literature. Speaking of them as "modalities" emphasizes the structural parallelism between these different forms of behavioral constraints.[18] The clearest statement of the approach is Lessig's:

> Now obviously, these four modalities do not regulate to the same degree—in some contexts, the most significant constraint may be law (an appeals court); in others, it is plainly not (a squash court). Nor do they regulate in the same way—law and norms, for example, typically regulate after the fact, while the market or architecture regulates more directly. The modalities differ both among themselves and within any particular modality. But however they differ, we can view them from this common perspective—from a single view from which we might account for (1) the different constraints that regulate an individual and (2) the substitutions among these constraints that might be possible.[19]

Taken together, Lessig's points imply that regulatory choices among modalities matter. A market may be cheaper than legal control; social norms may be cheaper still but run the risk of permitting hidden invidious discrimination. A good understanding of the characteristics of the available modalities therefore becomes essential to selecting a modality conducive to the goals of regulation, be they efficacy, cost reduction, fairness, expressive richness, administrability, risk minimization, or any of the other concerns that inform questions of policy. Recognizing them as distinct modalities invites comparison of the consequences of choosing one modality as opposed to another and focuses attention on their mutual influences.

B. *The Orthodox View: Software as Architecture*

"Software" does not appear on Lessig's list of four modalities, because he considers software to be a form of architecture. To understand the logic behind this claim, it is helpful to consider a related question. What does it

16. For a legal analysis of this assumption and an extensive bibliography, see Neal Kumar Katyal, *Architecture as Crime Control*, 111 YALE L.J. 1039 (2002).

17. LESSIG, *supra* note 1, at 92.

18. The first work that systematically analyzed different modalities as behavioral constraints was probably ROBERT C. ELLICKSON, ORDER WITHOUT LAW: HOW NEIGHBORS SETTLE DISPUTES (1991), which deploys a precise vocabulary of "controllers" and "rules" to talk about regulatory effects on behavior. Ellickson develops a theory of social norms and their relationship to law that has much in common with work in law and economics on the relationship between law and markets. *See id.* at 126-36.

19. Lessig, *supra* note 8, at 664.

mean to say that software "regulates" behavior, given that software runs only on computers while people live in the real world?

Lessig's idea of "dual presence" provides the necessary link. People who are using computers are in two places at once. They are online, in virtual spaces, and they are also offline, in the real world. Software constrains the behavior of their online selves; those constraints then have effects on the real people living in the real world.[20] Sometimes the connection is purely psychological. If I am insulted in a chat room because the chat room software did not let me block messages from my antagonist, or if I cannot reach the final level of a video game because I cannot defeat a sub-boss, then a software-imposed constraint has affected the happiness of an actual living person. Sometimes, the connection is even stronger, and important real-life interests and relationships are mediated by software systems. Online shopping, electronic voting, and file sharing come immediately to mind. Offline relations we regulate with law are directly affected by what happens online.

Put another way, just as we go into buildings and allow ourselves to be regulated by architecture, we also go into virtual spaces and allow ourselves to be regulated by them. The only difference is that we go "into" these virtual spaces virtually, by doing things on computers.

To many, this spatial metaphor has seemed conclusive. Just as the set of things we can physically do in physical spaces is determined by their physical architecture, the set of things we can virtually do in virtual spaces is determined by their virtual architecture—that is, by the software programs that create these spaces. What is more, physical architecture and software regulate their respective spaces in the same manner: automatically without human intervention, with immediate constraints rather than after-the-fact punishments.

Scholars have recognized problems with the spatial metaphor. First, "code is law," while an attractively pithy phrase, is a misleading way of saying that code is architecture.[21] Useful scholarship on the political economy of software's interactions with law has been forced into the awkward rhetorical posture of rejecting this seeming equation of software with law.[22]

20. *See* LESSIG, *supra* note 1, at 190.

21. *See* Cindy A. Cohn & James Grimmelmann, *Seven Ways in Which Code Equals Law (And One in Which It Does Not)*, *in* CODE: THE LANGUAGE OF OUR TIME 20 (Gerfried Stocker & Christine Schöpf eds., 2003), *available at* http://www.aec.at/en/archives/festival_archive/festival_catalogs/festival_artikel.asp?iProjectID=12315.

22. *See* R. Polk Wagner, *On Software Regulation*, 78 S. CAL. L. REV. 457 (2005) (discussing principles regulators should consider when deciding whether to regulate directly through law or by legislating particular software solutions); Tim Wu, *When Code Isn't Law*, 89 VA. L. REV. 679, 707-09 (2003) (describing software as a tool people use to help themselves evade legal regulation).

Second, although Lessig's idea of dual presence emphasizes the close connection between online and offline effects, many writers have been overly influenced by images of virtual "places." Some have even treated online conduct as somehow disembodied from any real-world nexus.[23] Scholars have intelligently critiqued the use of spatial metaphors to describe software-mediated activity.[24]

These two critiques challenge only the precision of the language used to describe software as a form of architecture. They do not contest the ultimate correctness of that description. But to think of software as "architectural" is to underestimate its power. Physical architecture, while powerful at times, is hard to deploy in a purposive manner. Even the most ambitious proposals for architectural regulation have modest goals compared with their expense. Programmers can articulate more detailed, more comprehensive, and far more complex systems of regulation than can any architectural regulators confined to working with the clumsy materials of the physical world.[25]

C. *A Better View: Software as Its Own Modality*

To see why software is similar to physical architecture but profoundly more powerful, we need to examine how software is written and used. The story begins with a programmer who has in mind some task she would like a computer to carry out. She attempts to envision as precisely as possible the details of the process by which she would like that task carried out. This precision is necessary because she must express her intention in the text of a computer program—a list of instructions, written in one of a number of artificial languages intelligible to a computer. Compared with human languages, these languages are highly constrained. Each of her instructions carries a fixed and precise meaning. Once she has finished editing the program, her role ends. She turns the program over to the computer on which it will run. As long as the computer is supplied with power and is free of mechanical and electronic glitches, it will read her instructions and

23. *See, e.g.*, John Perry Barlow, A Declaration of the Independence of Cyberspace (Feb. 8, 1996), *reprinted in* CRYPTO ANARCHY, CYBERSTATES, AND PIRATE UTOPIAS 27 (Peter Ludlow ed., 2001), *and available at* http://www.eff.org/~barlow/Declaration-Final.html.

24. *See, e.g.*, Dan Hunter, *Cyberspace as Place and the Tragedy of the Digital Anticommons*, 91 CAL. L. REV. 439 (2003); Orin Kerr, *The Problem of Perspective in Internet Law*, 91 GEO. L.J. 357 (2003); Timothy Wu, *When Law & the Internet First Met*, 3 GREEN BAG 2D 171 (2000).

25. None of this is to say that the analogy is always misleading. Many writers have used "architecture" in a more limited sense to refer to the system of institutional relationships and network connections that forms a backdrop to online behavior. When confined to this sense, the metaphor is more apt. *See, e.g.*, Mark A. Lemley & Lawrence Lessig, *The End of End-to-End: Preserving the Architecture of the Internet in the Broadband Era*, 48 UCLA L. REV. 925, 930-40 (2001); Daniel J. Solove, *Identity Theft, Privacy, and the Architecture of Vulnerability*, 54 HASTINGS L.J. 1227, 1238-43 (2003); Solum & Chung, *supra* note 6, at 823-49.

carry them out. A user may now encounter the program—that is, be regulated by it.

This encounter has an important dual quality. On the one hand, the software enables functionality. When I type in a chat room and my words appear on your screen, this communication has been enabled by the chat program, which in turn reflects the programmer's design decisions. Without her program, the communication could not happen.

On the other hand, software also limits behavior. By giving its users a set of possible actions, it excludes every action not within this set. Some of these exclusions are explicit: If you try to access a password-protected website without the proper credentials, the software on the web server will display a message telling you that such access is not allowed.[26] But some such limits are implicit: The software in a calculator program excludes word processing in the sense that text handling is simply not something that calculator software does. The calculator program both enables calculation and regulates it.

This description makes clear three defining characteristics of software: It is automated,[27] immediate,[28] and plastic. Because so much depends on these characteristics, it is worth dwelling briefly on each.

Software is automated. The only person needed to make software work is its programmer. Once she is done, the software can operate effectively on its own. Because software is automated, it can be an extraordinarily inexpensive way of solving problems. A computer costing $1000 can carry out perhaps a hundred million billion operations over the course of its useful life.[29] Once a piece of software has been written, the marginal cost of running it to handle another case can be vanishingly small.

Among other possible regulators, only physical architecture resembles software in being automated much of the time. Once a building is up, it tends to stay up; a locked door will deny a thief entry even if no guard is watching. In contrast, a court requires judges to hear cases, lawyers to argue them, and marshals to enforce the results. A market is the aggregation of the individual decisions made by its participants. A market in which these participants were uninvolved would be dysfunctional at best. Social norms, by their very definition, are social. Take away the people, and you take away the norms.

Software is immediate. Here is Lessig's description of the distinction between immediate (prospective) constraints and retrospective sanctions:

26. The password-protection example is Lessig's. Lawrence Lessig, *The Zones of Cyberspace*, 48 STAN. L. REV. 1403, 1408 (1996).

27. Lessig refers to this feature as "self-execution" or "agency." LESSIG, *supra* note 1, app. at 236-37.

28. Lessig also uses the term "present constraint." Lessig, *supra* note 2, at 510 n.32.

29. This figure assumes a computer with a three-gigahertz processor and a life of three years.

> Architecture and the market constrain up front; law and norms let you play first. For example, think of the constraints blocking your access to the air-conditioned home of a neighbor who is gone for the weekend. Law constrains you—if you break in, you will be trespassing. Norms constrain you as well—it's unneighborly to break into your neighbor's house. Both of these constraints, however, would be imposed on you *after* you broke into the house. They are prices you might have to pay later. The architectural constraint is the lock on the door—it blocks you *as you are trying* to enter the house. The market constrains your ownership of an air conditioner in the same way—it demands money before it will give you one.[30]

Again, the analogy between software and architecture holds. Think of the calculator and the password-protected Internet site. There is no need to punish someone who uses a calculator program to write a letter—it is simply not possible to use the program in that way.[31] Similarly, as long as the password-checking software on the site works, it constrains your access immediately.

Further, because software intervenes to constrain conduct prospectively, it must act on the basis of information available to it at the time of the conduct and on the basis of the program as the programmer wrote it before the conduct. Thus, in order for a class of information to be material to a decision made by software, the programmer must have specified the relevance of such data in the program itself. Software cannot—as law can—adapt its response in light of later-available information or a later determination that such information is relevant.[32]

Software is plastic. So far, Lessig's treatment of software as a species of architecture seems justified. But there is one crucial respect in which software profoundly violates expectations based on physical architecture: Software can make much more finely grained decisions. Frederick Brooks, in *The Mythical Man-Month*, probably the most influential book ever written on the process of software development, describes the profound degree of control a programmer has over her program:

30. LESSIG, *supra* note 1, app. at 237 (endnote omitted).

31. Technically, of course, if someone successfully hacks or evades the software, it has not acted as an effective constraint, and ex post sanctions may be needed to deter such conduct. My claim is not that software is always an effective constraint; in Section II.D, I discuss ways in which software may fail to constrain. The point here is that to the extent software constrains, it does so presently, in the here and now—exactly as physical architecture does, when one is not driving around the speed bumps or removing the filters from one's cigarettes.

32. Thus, the technical definition of "immediate" used in this Note implies its everyday meaning of "at once." A regulator that works through ex post sanctions can delay its response to a given action while it investigates and decides whether to intervene. On the other hand, a regulator that uses ex ante prohibitions must either allow or prohibit a given action at the moment that action is attempted.

> Finally, there is the delight of working in such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures.[33]

Whatever a programmer can express in a programming language, she can create. But what kinds of processes can be expressed in a programming language? Computer science's best answer to this question is the Church-Turing Thesis, which states that all moderately sophisticated programming languages are equally powerful, because each of them is capable of expressing everything any of the others could express.[34] It follows that what matters is whether the process we are looking at can be expressed in even one programming language; if so, then it can be expressed in any respectable programming language. In particular, anything about which we can be precise can be expressed as a computer program. If I can express my sense of what results should follow in a given set of circumstances in statements of the form "if *X* happens, then the result should be *Y*," I have actually written a computer program to carry them out.

I can be as general or as specific in my instructions as I want. If I am creating a virtual soda machine, I could have it emit a virtual can of soda when its virtual button is pushed. I could have it examine the complete history of the customer's interaction with virtual vending machines and only let it dispense a soda if the customer has never hit or shaken another vending machine. Or I could have it dispense its entire contents if the customer inserts a virtual dime between 3:55 and 4:00 a.m. on the second Wednesday of the month and pushes the "cola" button three times in less than eight seconds.

It is the combination of this precision with the immense leverage provided by automation that gives programmers such power. They can write programs that handle particular cases with any necessary level of detail while at the same time being able to handle those cases billions of

33. FREDERICK P. BROOKS, JR., THE MYTHICAL MAN-MONTH: ESSAYS ON SOFTWARE ENGINEERING 7 (anniversary ed. 1995).

34. *See* Alonzo Church, *An Unsolvable Problem of Elementary Number Theory*, 58 AM. J. MATHEMATICS 345, 356-58 (1936), *reprinted in* THE UNDECIDABLE: BASIC PAPERS ON UNDECIDABLE PROPOSITIONS, UNSOLVABLE PROBLEMS AND COMPUTABLE FUNCTIONS 88, 100-02 (Martin Davis ed., 1965) (defining "effective calculability"); A.M. Turing, *On Computable Numbers, with an Application to the* Entscheidungsproblem, 42 PROC. LONDON MATHEMATICS SOC'Y 230, 249-58 (1936), *reprinted in* THE UNDECIDABLE, *supra*, at 116, 135-44 (defining "computable"). Church's "effective calculability" and Turing's "computable" are provably equivalent, suggesting that both definitions capture the same, meaningful concept. *See* Turing, *supra*, at 149-51. For a readable modern discussion of the Church-Turing Thesis and its implications, see DOUGLAS R. HOFSTADTER, GÖDEL, ESCHER, BACH: AN ETERNAL GOLDEN BRAID 561-79 (twentieth-anniversary ed. 1999).

times. Physical architecture shares neither software's level of precision nor its nearly costless replication.

That programmers have such flexibility does not necessarily mean that users do. Our hypothetical programmer could easily choose to make her calculator program use decimal notation, scientific notation, or both. But once she has made that choice, the user cannot easily undo it. When users are powerless over software, it is often because programmers have made design decisions that leave users without power. Indeed, this imbalance is part of the effectiveness of regulation by software.[35]

II. FEATURES OF SOFTWARE AS A MODALITY OF REGULATION

This Part discusses the predictable consequences of choosing software as a regulator. These consequences follow from the basic characteristics of software identified in Part I.

A. *Software Acts According to Rules Rather than Standards*

The distinction between rule- and standard-based decisions is a staple of jurisprudential theory.[36] Rules provide the outcome for a set of cases in advance; standards presuppose case-by-case balancing. Rules state a basis for decisions independent of the decisionmaker; standards invite the decisionmaker to exercise discretion.

Because software is automated and immediate, it applies rules and not standards. The defining characteristic of rules is that they specify completely the results of cases in advance without leaving space for situation-specific discretion.[37] The very idea of a rule presupposes some sort of algorithm, some well-defined procedure that takes a set of inputs and produces a determinate result. When a programmer creates a program, she predetermines its responses to every possible input—to every possible "case" it may adjudicate. The algorithm is the rule.

Because software is automated, it operates without human discretion. On its way to making a decision, there is no place at which the result must be filtered through a human mind. Indeed, because software is immediate, there is not even a risk of discretion forcing itself into the process. In an ex post decision process, a party can always plead personal equities, extenuating circumstances, or the relevance of some other fact not specified

35. *See* RICHARD M. STALLMAN, *Why Software Should Be Free*, *in* FREE SOFTWARE, FREE SOCIETY: SELECTED ESSAYS OF RICHARD M. STALLMAN 119, 124-25 (Joshua Gay ed., 2002), *available at* http://www.gnu.org/philosophy/shouldbefree.html.

36. *See, e.g.*, Pierre Schlag, *Rules and Standards*, 33 UCLA L. REV. 379 (1985); Kathleen M. Sullivan, *The Supreme Court, 1991 Term—Foreword: The Justices of Rules and Standards*, 106 HARV. L. REV. 22 (1992).

37. *See* Cass R. Sunstein, *Problems with Rules*, 83 CAL. L. REV. 953, 961 (1995).

in the original rule. But as noted above, an immediate regulator closes the door to such pleas, because there is no way for a plea to affect the decisions that the regulator has already made. The only facts that can be relevant to a decision made by software are the facts the programmer considered relevant when she wrote the program.

Because software operates by means of rules and not standards, many jurisprudential observations about the nature of rules apply directly to software. Where we think that a rule is better than a standard, software will be an attractive regulator; where we prefer standards, software will be comparatively unattractive. For example, where we value consistency of application, we will tend to prefer rules to standards.[38] Conversely, where we believe we lack sufficient information to formulate good rules, we will tend to prefer standards.

At the same time, several of our expectations about rules do not apply to software. Most importantly, software rules achieve a perfection of ruleishness compared to which legal rules always fall short. A familiar critique of rules is that even facially precise rules invite a measure of discretion. In the application of a rule to particular facts, the characterization of the facts themselves will depend on the human decisionmaker's preconceptions and nonlegal sensibilities.[39]

But this critique has no bite against software rules, because software is automated all the way down. There is no separate fact-finding layer to provide human wiggle room. A software rule may be partisan—that is, it implements its programmer's idea of the "right" result, regardless of what anyone else may think—but that does not mean it is any less a rule. Even if the programmer herself changes her mind in response to the facts of a given case, it is too late. All she can do is alter the rule for future cases.

Significantly, we are accustomed to thinking that as the complexity of particularized rules increases, their formal realizability decreases. Increased complexity introduces ambiguities, unpredictable interactions between rules, and more opportunities to shade the "jurisdictional" facts that trigger a rule's application—all of which increase the discretion of the decisionmaker.[40] But where there is no discretion in the first place, complexity does not turn rules into standards: Software rules can become almost unimaginably complex without their hard edges blurring.

Further, rules enforced by software need not be capable of succinct articulation. When judges are the decisionmakers, a rule that cannot be written down cannot be read and applied. Even the famously prolix tax code

38. *See id*. at 974-77 (discussing several variations on this theme).

39. *See* Stanley Fish, *The Law Wishes To Have a Formal Existence*, *in* THE FATE OF LAW 159, 175-78 (Austin Sarat & Thomas R. Kearns eds., 1991); Sunstein, *supra* note 37, at 985.

40. *See* Duncan Kennedy, *Form and Substance in Private Law Adjudication*, 89 HARV. L. REV. 1685, 1690 (1976); Sunstein, *supra* note 37, at 984-85.

presents a set of rules that are simple enough for humans to read and apply to particular facts. But the rule implemented by a program is, ultimately, the totality of the program itself; there need not be any version that a person could explain in her own words.[41]

Consider Deep Blue, the computer program that is now the word's top-ranked chess player. The programmers who created it are worse chess players than their program, which uses a combination of massive trial and error and a highly abstracted evaluation of the quality of a chess position.[42] In choosing any given move, Deep Blue makes millions of individual decisions, any of which may be a but-for cause of the move it ultimately makes. Deep Blue applies a rule in the sense that its response to a given board position is completely predetermined by its program, but there is no person alive who could apply that rule in any way other than by simulating the millions of steps Deep Blue takes on its route to a decision. A person carrying out this process of rote computation will not gain much insight into why Deep Blue selects one move over another, any more than a person studying an MRI of a philosopher's brain will learn something about philosophy. Predicting Deep Blue's moves from outside is harder than playing grandmaster-level chess—too hard, in other words, for any person to do. Indeed, not even a program's authors may fully understand the rule the program follows. Deep Blue's programmers certainly do not.

The creation of functional rules that cannot be explained is made possible by the perfect formal realizability of software rules. Software can successfully apply rules whose complexity would make them collapse under their own weight if humans were forced to apply them.

B. *Software Need Not Be Transparent*

Among the virtues associated with the rule of law are at least two we usually think of under the heading of "transparency." First, good law is predictable. One should be able to predict the application of a law and bring one's conduct into compliance with it. Second, good law is accountable. One should be able to see who is responsible for a law, so that only those who will stand behind their actions exercise authority.

41. The single most important result of computer science, the Turing Theorem (not to be confused with the Church-Turing Thesis, although they are related), says that there is no general way to find out much about computer programs. (More precisely, it is impossible in general to tell whether a given computer program will ever finish its work. *See* Turing, *supra* note 34, at 132-34.)

42. FENG-HSIUNG HSU, BEHIND DEEP BLUE: BUILDING THE COMPUTER THAT DEFEATED THE WORLD CHESS CHAMPION 24, 197-203 (2002).

It is sometimes asserted that rules are well correlated with these twin virtues of transparency.[43] While this is generally the case for legal rules, we have long known that ruleishness per se is analytically independent from these transparency values.[44] And software can deliver ruleishness without guaranteeing transparency. As the above discussion of Deep Blue suggests, it is possible to have a well-functioning software system whose reasoning is inscrutable even to its creators. More troublingly, this possibility is not confined to cases of raw complexity. Software is capable of applying even quite simple rules in a way that obscures both the pattern of their effects and their source.[45]

To understand software's transparency deficit, it is helpful to start by considering how the other modalities stack up. Law is in many ways the paragon of openness: The phrase "rule of law" is nearly synonymous with transparency in adjudication.[46] Because law is articulated and applied by judges who must explain their conclusions in every case, the system has built in a great deal of accountability in the form of transparent reasoning. A judicial opinion foregrounds both the specific rule of decision being applied and the institutional source of that rule.

Markets and norms both depend on predictability. Social norms, by their nature, require individuals to internalize the relevant group's sense of acceptable and unacceptable behavior,[47] while markets depend on price signals to function.[48] Where norms and markets sometimes fall short is on the accountability side of transparency. Their decentralized nature means that it is hard to locate any one agent responsible for the overall constraint being imposed. This lack of agency is sometimes considered a good thing, because it enables social groups to respond informally to their members' interests and it accounts for the seemingly uncanny ability of markets to aggregate vast quantities of information—the so-called "invisible hand." But it can also be troubling. Destructive or invidious social norms can be difficult to uproot because there is no single agent with the power to alter them,[49] and markets have been frequently criticized for their disembodiment of aggregate controls from individual choices.

43. *See* Kennedy, *supra* note 40, at 1688; Antonin Scalia, *The Rule of Law as a Law of Rules*, 56 U. CHI. L. REV. 1175, 1178-81 (1989).

44. *See* Kennedy, *supra* note 40, at 1701-02; *see also* LON L. FULLER, THE MORALITY OF LAW 39 (2d ed. 1969).

45. *Cf.* Wagner, *supra* note 22, at 479-80 (treating transparency as a safety valve on regulation).

46. *See* Scalia, *supra* note 43, at 1187.

47. *See* LESSIG, *supra* note 1, app. at 237-38.

48. *See* James Boyle, *A Theory of Law and Information: Copyright, Spleens, Blackmail, and Insider Trading*, 80 CAL. L. REV. 1413, 1443-45 (1992).

49. *See* Dan M. Kahan, *Gentle Nudges vs. Hard Shoves: Solving the Sticky Norms Problem*, 67 U. CHI. L. REV. 607 (2000).

Physical architecture is more ambiguous. It is certainly predictable in the sense that an object exists or it does not, and its basic physics are intuitive. Speed bumps punish those who drive too fast. Similarly, because regulation by physical architecture is so inflexible, there will be few examples of cases in which it regulates in a way that is not predictable to those being regulated but is purposeful from the point of view of the architect. Walls, streets, and windows are not very selective; their rules of behavior are written in very broad terms. On the other hand, physical architecture can conceal the design choices embedded in it. Lee Tien gives the example of doors on phone booths: Their presence increases one's expectation of privacy for Fourth Amendment purposes, but few would look closely at the regulatory implications for privacy of a government mandate that phone booths not have doors.[50] To the extent that physical architecture with a regulatory agenda figures in our lives as a given, it is not always transparent.

In one sense, it is an immediate corollary of concerns noted above that software can be unpredictable to those regulated by it. A software rule that is so complex as to defy human understanding and explanation does not provide predictability. The rule itself is unambiguous as applied to any particular set of facts, but prospectively, there may be no simpler way to predict whether a given action is allowed than by attempting to engage in it.[51]

This problem of unpredictability can arise even when the underlying rule is not in itself excessively complex. Software is asymmetric. The programmer can determine its responses, but the user sees only the results of the software's individual decisions—allow this action, forbid that one—and lacks access to accurate knowledge of the set of inputs that determined a particular output.[52] All the user knows is that a particular action is disallowed, without necessarily understanding why. In particular, because users are confined to dealing with software on a case-by-case basis, it may be very difficult to detect systemic discrimination where unpredictable software is involved.[53]

50. Lee Tien, *Architectural Regulation and the Evolution of Social Norms*, INT'L J. COMM. L. & POL'Y, Autumn 2004, at 1, 4-5, http://www.digital-law.net/IJCLP/Cy_2004/pdf/Lee_Tien_ijclp-paper.pdf.

51. Computational learning theory has shown that under many circumstances, it is infeasible to derive a general rule merely by observing the results the rule dictates in particular cases. *See* MICHAEL J. KEARNS & UMESH V. VAZIRANI, AN INTRODUCTION TO COMPUTATIONAL LEARNING THEORY 123-42 (1994).

52. Computer science has drawn an important distinction between the existence of a solution to a problem and the feasibility of determining that solution. *See* CHRISTOS H. PAPADIMITRIOU, COMPUTATIONAL COMPLEXITY 230-34 (1994). Further, it is often possible to prove that one has carried out a computation without revealing the computation itself. *See* ODED GOLDREICH, FOUNDATIONS OF CRYPTOGRAPHY: BASIC TOOLS 184-90 (2001).

53. *See* Batya Friedman & Helen Nissenbaum, *Bias in Computer Systems*, 14 ACM TRANSACTIONS ON INFO. SYS. 330, 331 (1996).

If software were fully accountable, this issue might not be as troubling. But two features of software undermine its accountability. The first it shares with physical architecture: It may not occur to those regulated by software to think of the man behind the curtain, to conceive of a restrictive design decision as being a decision at all. Most streaming media players do not give users the ability to save the stream to a local hard drive, a design decision made quite deliberately, usually to advance the interests of copyright holders.[54] Users may not realize that this decision was made deliberately.

The second feature that undermines software's accountability is also characteristic of markets and norms. In a complex software system, it may be nearly impossible to determine who made the relevant regulatory decision. In any software system that is itself composed of smaller subsystems, the actual basis for decision may well be a composite of rules specified by different programmers. If I try to load a web page and receive a "not found" error, the request may have failed because I mistyped it, because my computer is not connected to the Internet, because of a failure at any number of places between my computer and the web server, because the web server is unavailable, or because the page itself is no longer on the server. Without further inspection, I cannot easily identify the reason for the failure and thus cannot easily determine who is responsible for it.[55]

Further, even a regulator who wishes to use software transparently will face difficulty, both because software has a hard time offering contemporaneous explanations of its decisions and because even the programmer herself may not be able to explain particular decisions. The most basic regulatory strategy to combat opacity is explanation, in which the regulator says what it is she is doing as she makes a decision. The effectiveness of this strategy can depend significantly on one's ability to verify that her explanation is accurate. As anyone who has ever dealt with cryptic error dialog boxes can testify, however, software does not always do a very good job explaining itself, even when its programmer has tried to offer such explanations.[56]

Generations of artificial intelligence researchers can attest that computers remain unable to engage in sustained dialogue with users.

54. *Cf.* RealNetworks v. Streambox, No. C99-2070P, 2000 U.S. Dist. LEXIS 1889, at *3-13 (W.D. Wash. Jan. 18, 2000) (holding that a program enabling users to save streaming media files to a hard drive violates the Digital Millennium Copyright Act).

55. *See generally* Helen Nissenbaum, *Accountability in a Computerized Society*, 2 SCI. & ENGINEERING ETHICS 25 (1996), *available at* http://www.nyu.edu/projects/nissenbaum/papers/accountability.pdf.

56. *See* AJ Cunningham, C Compiler Errors (For Real), http://www.netfunny.com/rhf/jokes/91q3/cerrors.html (last visited Feb. 12, 2005); Funny Error Messages and Strange Bugs, http://home.swipnet.se/moose/funerr.htm (last visited Feb. 12, 2005); Snopes.com, Urban Legends Reference Pages: Computers (Haiku Error Messages), http://www.snopes.com/computer/internet/haiku.asp (last visited Feb. 12, 2005).

Computers have great difficulty parsing natural human language sufficiently well to be able to formulate appropriately responsive replies.[57] Thus, they can only offer whatever explanation the programmer was able to draft in advance. It is hard to adapt explanations to the concerns of a particular user. To take a trivial example, the majority of Internet sites are monolingual, and only a small fraction are properly accessible to blind users.[58] In another vein, many skilled computer users find themselves frustrated by automated software technical support aimed at novices. The desire to "speak to a person who understands" is a strong one. Dialogue is a process value; the opportunity to be heard can make the outcome more intelligible.

Sometimes programmers can create such dialogue by writing good help text and having people who understand why the software makes the decisions it does provide technical support. But recall the discussion of Deep Blue. Programmers who do not themselves understand their program's reasoning may not be able to provide such support. Imagine Deep Blue's programmers hauled into court and ordered to justify its decision to sacrifice a pawn. Their inability to explain is confirmation of the practical unpredictability of their program.[59]

Another way of understanding why software is so often unpredictable is to note that programming is a difficult profession. Programming absorbs the attention of hundreds of thousands of intelligent, trained professionals. All of them spend substantial amounts of time debugging their code. Almost every computer program makes decisions that its creator neither expects nor intends nor understands. The ambition of software engineering is to remove the most important bugs and reduce the unpredictable errors (i.e., "wrong" decisions) to a tolerable level, not to eliminate them entirely.[60]

C. *Software Rules Cannot Be Ignored*

Because software is automated, its rules can apply even in realms where it would be extremely hard for law to reach. Even though I might want to

57. *See, e.g.*, Stuart M. Shieber, *Lessons from a Restricted Turing Test*, 37 COMM. ACM 70, 71-74 (1994).

58. DISABILITY RIGHTS COMM'N, THE WEB: ACCESS AND INCLUSION FOR DISABLED PEOPLE 9 (2004), http://www.drc-gb.org/publicationsandreports/2.pdf ("Most websites (81%) fail to satisfy even the most basic Web Accessibility Initiative category."); *see also* Pariah Burke, 81% of Websites Inaccessible to Disabled, So Is the Report That Says So, http://iampariah.com/blog/2004/04/81-of-websites-inaccessible-to-disabled-so-is-the-report-that-says-so/ (April 21, 2004, 00:50 PST) (noting that the PDF version of the Disability Rights Commission's own report is similarly inaccessible).

59. Indeed, its unpredictability is in some sense necessary to its operation because otherwise its opponents could train against it too easily.

60. *See* ROGER S. PRESSMAN, SOFTWARE ENGINEERING: A PRACTITIONER'S APPROACH 212-16 (5th ed. 2001).

duplicate my copy of Microsoft Office XP for a friend, I cannot: Microsoft's "product activation" feature ties my copy to my computer.[61] Sometimes this degree of control may be unjustified as a matter of social policy,[62] and sometimes it may apply in situations the regulator would rather leave untouched.

Scholarship on regulatory modalities has convincingly demonstrated that it is often good social policy, rational, and efficient for people to act with indifference to legal rules.[63] Law, extended into such realms, disrupts efficient social norms in those cases in which it is applied. Because software can reach so many more transactions than can law, it can disrupt more cases. And because software, unlike law, is immediate, the effects of each disruption are more severe.

Begin with the Coase Theorem: The most important determinant of the efficiency of a legal rule is the structure of transaction costs. An efficient legal rule minimizes the transaction costs people pay on their way from the initial distribution of entitlements to a Pareto-efficient distribution. The usual phrase for such transactions is "bargaining in the shadow of the law,"[64] and the model it implies is clear. People take as given the legal default rules governing their situation, and where those rules produce inefficient distributions, they strike individual bargains for more efficient outcomes.

In reality, however, the law is neither as important nor as powerful as this model presumes. Robert Ellickson has claimed that the presence of shared social norms enables groups to self-regulate informally.[65] Not only is the law's default rule not the starting place for their negotiations, they do not need to know what law governs their situation or even know that it is covered by law. If they have an effective norm-enforced default rule to situate an entitlement, they will not need to turn to law.

This displacement works because law is neither automated nor immediate. Because law is not automated, if the parties to a potential dispute are satisfied with its resolution, the law will leave them alone. And because law is not immediate, it is content to wait until one of the parties comes before it with a complaint. It leaves time for them to strike an informal deal with each other, or for internalized community norms to keep a dispute from arising. The law does not come running in and attempt to

61. Microsoft Corp., Windows XP Product Activation, http://www.microsoft.com/windowsxp/evaluation/features/activation.mspx (last edited Aug. 29, 2002).

62. *See* Boyle, *supra* note 5, at 196-200; Julie E. Cohen, Lochner *in Cyberspace: The New Economic Orthodoxy of "Rights Management,"* 97 MICH. L. REV. 462, 468-80 (1998).

63. *See* ELLICKSON, *supra* note 18, at 280-83.

64. This phrase comes from Robert H. Mnookin & Lewis Kornhauser, *Bargaining in the Shadow of the Law: The Case of Divorce*, 88 YALE L.J. 950 (1979).

65. ELLICKSON, *supra* note 18, at 167-83.

hand physical possession of a whale to the whaler who first harpoons it; it is content to adjudicate legal title to the whale later on.[66]

But consider what would happen if the law could interpose itself directly into people's dealings. If law were automated, parties would need to sign off explicitly on every readjustment of resources between them. Worse, if law were immediate—if its decisions about entitlements translated directly into physical reality—parties would have to go through the cumbersome process of formally exchanging possession. So long as the legal rule matched the content of their informal norm, no damage would be done, but as soon as the law deviated from what they would have chosen in its absence, the transaction costs would begin to mount.

This is exactly what happens when people are regulated by software, which can and does impose itself on their dealings. If the resource they are interested in is a virtual one—a domain name, an item inside a multiplayer game—software determines at the outset who is entitled to it and who is not. If the software chooses "wrongly," it is up to the parties to fix the mistake.

Now, it is quite possible for software to get many, indeed, most cases right. But the law and economics literature is rife with examples showing that essentially any explicit ex ante rule—and rules are what software applies—will be inefficient in some cases that social norms among the relevant community could get right.[67] In these cases, software will impose transaction costs that law would not.

In this respect, software is like physical architecture, like laws of nature, like the world as we find it. But unlike these other constraints, software can be used instrumentally—to settle the distribution of resources. These other immediate regulators are simply not plastic enough for this possibility to arise. Because there is no question of substantially changing the way they distribute entitlements, the inefficiencies they generate are not open to real evasion, nor can we turn such regulators "off" in the same way. The best we can do is find ways to transact around them, and so we do. With software, these added costs are the consequence of a decision over which someone has control, and so the possibility of trading them off against the benefits conferred by software-based regulation can arise.

Transaction costs are by definition context dependent. The effects on transaction costs of using software are also context dependent. The observation that software-imposed rules must be transacted around rather

66. *See id.* at 191-206 (discussing legal and nonlegal rules governing property in whales).

67. Ellickson's discussion of whaling customs provides a good example. The community of whalers employed at least three different customs for determining which ship had the right to a whale, and the custom chosen depended on the type of whale, local geographic conditions, and the hunting technology in use. *Id.*

than ignored plays out very differently in different regulatory contexts. Here are three examples.

In some settings, the regulator is interested in helping the users collaborate to their mutual advantage and is indifferent about the details as long as the users are collectively happy. This is the situation in many computer games, where the users' goal is typically to have fun together, and in many productivity applications, where the users' goal is typically to collaborate in preparing a document. In such settings, it often makes sense for the regulator both to optimize software rules so that they track the most common cases and to provide users with easy software-based mechanisms to strike corrective bargains. These software rules resemble legal default rules.

In other settings, the interests of the users are more adversarial, but the regulator is still unconcerned with the details of their interaction so long as they are happy and they use the software. This is the pattern of many online marketplaces, whose goal is to provide users with an automation-fueled reduction in transaction costs.[68] These software rules resemble perfectly enforced mandatory legal rules. If you enter the system at all, you will be bound by them.

In yet other settings—in many DRM systems, for example[69]—regulators who use software are concerned that the software be the exclusive medium for negotiating exchanges between users and, further, that only certain kinds of exchanges be allowed. Unlike in the previous case, however, regulators cannot rely on the users' self-interest to prevent attempts to opt out of the distribution of entitlements set up by the software. In this case, as in the previous one, the efficiency consequences of an inappropriate software rule will be magnified by users' inability to contract around it.

Finally, note that because software operates automatically, the programmer may not be aware that her software's rules are operating inefficiently. All she can tell is that the rule is being applied, not how people feel about the result. The natural tendencies that make other regulators aware of their inefficiencies and pressure them to fix those inefficiencies are absent with software. Markets will respond with price changes; judges will respond with equitable exceptions; norms are notoriously adaptable. Software is comparatively oblivious to its own inefficiencies.

68. *See infra* Section III.A.
69. *See infra* Section III.B.

D. *Software Is Vulnerable to Sudden Failure*

Software failures range from the huge and newsworthy—a software bug caused the eighteen-million-dollar Mariner I probe to go off course shortly after launch in 1962[70]—to the ordinary and everyday.[71] Software gives us the Internet; software failures have nearly taken it away.[72] Software is always, in a sense, on the brink of failure.

Specifically, software is vulnerable to failure in three related ways: It is *buggy*, it is *hackable*, and it is *not robust*. Software's intrinsic bugginess is an emergent property of its complexity and has already been discussed. The next two Subsections discuss hackability and robustness in turn.

1. *Software Is Hackable*

First, everything noted above about the power of software applies not only to its intended uses but also to uses made of it by malicious hackers. Thus, if Program *X* regulates some activity, a hacker who succeeds in replacing Program *X* with Program *X′* of her own devising will have gained the same absolute control over that activity that the original programmer once enjoyed.

Second, everything noted above about the endemic bugginess of software applies with particular force to software's "security" features—those parts of a system designed to keep a hacker from taking control. Errors as slight as a programmer's use of "`x = y`" instead of "`x == y`" have caused many security holes in operating systems.[73] Such seemingly small differences between two computer programs can have wildly disproportionate consequences. This difference could be the point of attack of a worm program, one that tricks a computer into following the program's instructions instead of those of the computer's owner. Because eliminating bugs from software is so difficult, it is also difficult to secure a complex computer system. Hackers can often gain significant toeholds even in mature and heavily guarded systems.[74]

70. Gladwin Hill, *For Want of Hyphen Venus Rocket Is Lost*, N.Y. TIMES, July 28, 1962, at A1.

71. An audit conducted by Earthlink, for example, found that the nearly five million computers analyzed contained over twenty-five million instances of "spyware" and "adware," software that users almost certainly had not voluntarily chosen to allow on their computers. EarthLink, Earthlink Spy Audit, http://www.earthlink.net/spyaudit/press (last visited Feb. 23, 2005).

72. *See* KATIE HAFNER & JOHN MARKOFF, CYBERPUNK: OUTLAWS AND HACKERS ON THE COMPUTER FRONTIER 251-341 (1991) (discussing the Internet worm accidentally released by Robert Tappan Morris).

73. In the C programming language, "`x = y`" sets `x` to be equal to `y`, whereas "`x == y`" is a test to see whether `x` and `y` are already equal.

74. *See generally* BRUCE SCHNEIER, SECRETS AND LIES: DIGITAL SECURITY IN A NETWORKED WORLD (2000).

Third, and crucially, hackers get to use software, too. The simplest form of such counterprogramming is to write one program that wages war on another program. The computer world is awash in brute force attacks. People guessing passwords use "dictionary attacks" that guess every possible password, one at a time. Against some classes of Windows computers, for example, a dictionary attack can successfully guess a password in seconds.[75] A related technique is to try the same attack against millions of different computers. The Blaster worm, for example, would randomly attempt to contact other computers on the Internet and would try the same attack against each. It may have infected more than one million computers.[76] Software automates the process of breaking other software, making such attacks far more likely to succeed.

Another class of software-abetted trickery involves leaving a program itself alone but embedding it in a larger system whose overall effects are contrary to the intent of the original programmer.[77] A DRM system may use strong encryption to prevent eavesdropping by nonpaying third parties during music distribution, for example. But that same strong encryption means that the DRM system could be a useful component for file sharers: They could use it to trade files in a way that is secure against eavesdropping by third-party copyright holders. The DRM system is doing exactly what it was programmed to do—keep people not party to a communication from learning anything about what is being said—but in a way that would be considered a serious failure by the people who programmed it.

Even more creatively, users determined to evade a particular restriction of software can often just recreate the software system and delete the restriction they dislike. They copy, reverse engineer, or independently recreate the program and make whatever changes they desire. In the 1980s, when most software came on floppy disks with copy protection software, stripping the copy protection from commercial software was one of the most popular techniques by which hackers demonstrated their skill.[78]

75. Robert Lemos, *Cracking Windows Passwords in Seconds*, ZDNET NEWS, July 22, 2003, http://news.zdnet.com/2100-1009_22-5053063.html.

76. Ellen Messmer, *Blaster Worm Racks Up Victims*, PC WORLD, Aug. 15, 2003, http://www.pcworld.com/news/article/0,aid,112047,00.asp.

77. The possibility of embedding one software system inside another is a fundamental technique of programmers. Entire software architectures are designed to allow later programmers to embed software written by earlier ones in contexts not explicitly anticipated by those earlier programmers. *See, e.g.*, PRESSMAN, *supra* note 60, at 721-22.

78. For a collection of hundreds of screenshots of cracked Apple II games, see Artscene, Apple II Crack Screens, http://artscene.textfiles.com/intros/APPLEII/.thumbs.html (last visited Feb. 12, 2005).

2. *Software Is Not Robust*

Because software is immediate and automated, it can fail almost instantaneously and without raising alarms among people who could otherwise intervene.[79]

Start with the question of time. It takes years to learn whether a legal loophole really works. Even if you find a rule exactly on point, you might still be stopped by an equitable exception or a jurisdictional nicety. Other regulators—markets, norms, physical architecture—are also not easy to probe quickly. It takes time to try every doorknob on a block; it takes even longer to have enough conversations with people to feel confident that they will not condemn you for doing what you contemplate doing. Especially if you don't have a particular exploit already in mind, finding one that works is a slow and tedious process. Software, by contrast, is immediate. You can try out a possible exploit, and the software will tell you right away whether or not you've succeeded. If not, you—or rather, your computer—have wasted at most a few milliseconds.[80]

More seriously, once a software system has been hacked or has failed of its own accord, the lack of human review means that it will not be able to recover on its own. The legal system is filled with people whose job is to prevent loopholing, including judges, opposing counsel, prosecutors, and police. If a specific result is discontinuous with the surrounding doctrine or runs against broader policy values, these monitors will find a way to stop it or attempt to bring the issue to the attention of others with the power to close the loophole. The legal system's technique of reasoned case-by-case adjudication is designed to let it fix discontinuities as they are observed, to help it spot end runs and determine whether they should be allowed.

Architecture, markets, and norms also display features that provide robustness. An important aspect of physical architecture is visibility.[81] Regulation by architecture uses visibility in its idea of natural surveillance—that good architecture is designed so that the conduct of potential intruders can easily be seen by many residents and so that the intruders are aware their actions can be seen.[82] Precisely because markets

79. *Cf.* Wagner, *supra* note 25, at 480 ("[S]oftware regulation can 'fly under the radar,' avoiding the oversight, both formal and informal, that occurs in even the least interventionist forms of legal regulation . . . .").

80. *Cf.* SCHNEIER, *supra* note 74, at 18-19 ("Fast automation makes attacks with a minimal rate of return profitable.").

81. *See generally* MICHAEL SORKIN, LOCAL CODE: THE CONSTITUTION OF A CITY AT 42° N LATITUDE 30-31 (1993) (treating views as a basic architectural element).

82. Katyal, *supra* note 16, at 1050. Katyal has attempted to extend this notion to cyberspace. Neal Kumar Katyal, *Digital Architecture as Crime Control*, 112 YALE L.J. 2261, 2264 (2003). Notice the difference between asking software to make actions visible to others and asking software to make decisions itself. In the real world, architecture that attempts to do this security job by itself has been notoriously ineffective—indeed, the harshness of visible security measures

function as devices for collecting and distributing information, they encourage participants to seek out relevant information, including about the behavior of other participants. Arbitrage makes markets hum, but it also helps to close the price gaps it exploits. And as for norms, they are enforced, indeed constituted, by people. If you do something without hiding it (and no regulator can touch conduct of which it is ignorant), then by definition it has come to the attention of the people who might regulate it by norms.

But software is automated. A person doing a page of calculations who hits a division by zero will know that something has gone wrong and check her earlier work to find the mistake. A computer program hitting the same division by zero will simply crash. Even when software is backed up by human eyes, it has a very difficult time figuring out what has gone—or is about to go—wrong. Debugging is the most difficult, tedious, and annoying part of programming. Forensic debugging—figuring out the cause of a fault after that fault has occurred—is even more difficult.[83] It is generally considered at least ten times more expensive to fix a bug found after release than before.[84]

Even worse, when the problem is not something so obvious as a crash, it is not always clear what the human supervisors ought to be watching for. If the software is regulating as intended, it will likely be producing enormous quantities of data.[85] The words programmers use to describe such data and the process of examining it—for example, one "grovels" through a "dump"[86]—give an idea of the nature of the task. It is not a hunt for a needle in a haystack. It is a hunt for the broken needle in a field covered two feet deep in other needles.

III. CASE STUDIES

The four predictable consequences of regulation by software identified in the previous Part—ruleishness, opacity, ubiquity, and fragility—provide a methodology for evaluating the use of software in particular regulatory contexts. First, one sketches out the key features of the problem domain and

tends to make architecture lifeless and antisocial, inadvertently reinforcing norms that such spaces are inhumane and need not be treated with respect.

83. For examples of forensic debugging techniques, see Paul Richards & Jörg Wunsch, *Kernel Debugging*, *in* FREEBSD DEVELOPERS' HANDBOOK (2004), http://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/kerneldebug.html.

84. *See* BARRY W. BOEHM, SOFTWARE ENGINEERING ECONOMICS 40 (1981).

85. The total amount of recorded information produced in 2002 is estimated to be about five exabytes, an amount equal to about 800 megabytes per person worldwide. PETER LYMAN & HAL R. VARIAN, HOW MUCH INFORMATION? 2003, at 1 (2003), http://www.sims.berkeley.edu/research/projects/how-much-info-2003/printable_report.pdf.

86. *See* THE NEW HACKER'S DICTIONARY 165-66, 228-29 (Eric S. Raymond ed., 3d ed. 1996), *available at* http://www.catb.org/~esr/jargon/.

the possible software solution. One then runs through the list of predictions to see how these common features of regulation by software are likely to emerge in the case at hand.

The next step is to ask whether these expected results are consistent with one's regulatory goals. If discretion is unnecessary, the need for transparency low, the potential additional transaction costs small, and the risk of software failure untroubling, then regulation by software is a good fit to the task at hand. If, on the other hand, one or more of these conditions do not apply, the shape of the inquiry shifts. The question then becomes whether one can take steps to mitigate the undesirable consequences of choosing software—with the costs and consequences of those steps factored into one's overall normative evaluation of whether software is the best choice among modalities.

Pervasive regulation by software either is already taking place or has been proposed for many policy domains. I discuss two: online markets and DRM systems. I could equally well have discussed privacy,[87] electronic voting,[88] virtual property,[89] access to computer systems,[90] peer-to-peer file sharing,[91] FCC regulation of telecommunications,[92] the control of offensive online speech,[93] or many others. The goal of this Part is not to advance the state of discussion of any of these areas. Instead, by showing that the regulation-by-software model speaks intelligibly to existing discussions in disparate areas, I offer evidence that it is both correct and generally applicable.

A. *Online Markets*

The first example is software-mediated markets—marketplaces in which the interactions of buyers and sellers are controlled by software. Participants interact with the marketplace through a layer of software, which determines when and on what terms transactions take place.

87. *See, e.g.*, Jonathan Zittrain, *What the Publisher Can Teach the Patient: Intellectual Property and Privacy in an Era of Trusted Privication*, 52 STAN. L. REV. 1201, 1203 (2000) (relating "trusted systems" (i.e., DRM) to the management of privacy online).

88. *See, e.g.*, ERIC A. FISCHER, CONG. RESEARCH SERV., ORDER CODE RL32139, ELECTION REFORM AND ELECTRONIC VOTING SYSTEMS (DREs): ANALYSIS OF SECURITY ISSUES (2003).

89. *See, e.g.*, F. Gregory Lastowka & Dan Hunter, *The Laws of the Virtual Worlds*, 92 CAL. L. REV. 1 (2004).

90. *See, e.g.*, Katyal, *supra* note 82; Eric J. Feigin, Note, *Architecture of Consent: Internet Protocols and Their Legal Implications*, 56 STAN. L. REV. 901 (2004).

91. *See, e.g.*, Lior Jacob Strahilevitz, *Charismatic Code, Social Norms, and the Emergence of Cooperation on the File-Swapping Networks*, 89 VA. L. REV. 505 (2003).

92. *See, e.g.*, Crawford, *supra* note 5; Kevin Werbach, *Supercommons: Toward a Unified Theory of Wireless Communication*, 82 TEX. L. REV. 863 (2004).

93. *See, e.g.*, United States v. Am. Library Ass'n, 539 U.S. 194 (2003); Lawrence Lessig, *What Things Regulate Speech: CDA 2.0 vs. Filtering*, 38 JURIMETRICS J. 629 (1998).

Typically, these transactions are sales; the software matches buyers and sellers and assists them in negotiating a price.

This description encompasses a striking range of goods and transactional models. Amazon zShops allows sellers to set prices for and post descriptions of their items; buyers can search to find all the sellers who are offering a particular item.[94] eBay similarly uses seller-posted descriptions but sets prices by auction.[95] Priceline, which matches travelers with travel services, uses a reverse auction, in which buyers name a price and the system searches for sellers willing to meet that price.[96] The NASDAQ stock exchange is entirely computerized. Its software takes bids from both buyers and sellers and finds market-clearing prices.[97]

In all of these markets, software mediates the transactions. Norms and law are also at work. The underlying resources, whether they are tangible personalty, contracts of carriage, or SEC-listed securities, are all subjects of legal regulation. Once the software determines that a deal has been struck, both buyer and seller find themselves parties to a legally enforceable contract.[98] The software here constrains participants to enter only into certain kinds of legal relationships, and only in certain ways. There is no mechanism on eBay for a consumer to attach a list of side conditions to her bid; payment on an Amazon zShops item must go through Amazon's payment service.[99]

The companies building such markets have, in many cases, been fabulously successful[100] because these markets have been fabulously successful in attracting participants.[101] This success is not accidental: Mediating online markets is an excellent use of software's regulatory powers.

94. Amazon.com, zShops Features, http://www.amazon.com/exec/obidos/tg/browse/-/537856/ref=br_bx_c_2_0/103-0524332-2241431/104-3515892-4811160? (last visited Feb. 12, 2005).

95. eBay, Getting Started, http://pages.ebay.com/help/newtoebay/getting-started.html (last visited Feb. 23, 2005).

96. Priceline.com, Two Great Ways To Save on the Airline Tickets You Want!, http://www.priceline.com/customerservice/faq/howitworks/air.asp (last visited Feb. 23, 2005).

97. SEC, Nasdaq, http://www.sec.gov/answers/nasdaq.htm (last visited Feb. 23, 2005).

98. *See, e.g.*, eBay, Your User Agreement § 4, http://pages.ebay.com/help/policies/user-agreement.html (last visited Feb. 23, 2005) ("By bidding on an item you agree to be bound by the conditions of sale included in the item's description so long as those conditions of sale are not in violation of this Agreement or unlawful.").

99. Amazon.com, Participation Agreement, http://www.amazon.com/exec/obidos/tg/browse/-/537790/104-3515892-4811160 (last visited Feb. 23, 2005).

100. For example, as of this writing, Amazon has a market capitalization of $13 billion. Yahoo!, Summary for Amazon.com Inc., http://finance.yahoo.com/q?s=AMZN (last visited Apr. 27, 2005). eBay has a market capitalization of $42 billion. Yahoo!, Summary for eBay Inc., http://finance.yahoo.com/q?s=EBAY (last visited Apr. 27, 2005).

101. eBay has more than 100 million registered users. Press Release, eBay, eBay Inc. Announces Fourth Quarter and Full Year 2004 Financial Results (Jan. 19, 2005), *available at* http://investor.ebay.com/news/Q404/EBAY0119-777666.pdf.

1. *Online Markets Rely on Ruleishness*

Setting ground rules for market transactions is perhaps the prototypical example of a situation in which we prefer rules to standards. A generation's worth of law and economics scholarship on contract law has argued that economic efficiency is maximized when the rules for contracting are clear and clearly understood. Decision by rule reduces both the ex ante information costs associated with planning transactions and the ex post transaction costs of relying on the system to resolve disputes.[102]

In markets where the goods are highly standardized, such as stock exchanges, the use of highly automated systems has been driving costs down dramatically.[103] Even where the underlying goods being exchanged are nonhomogeneous, as on eBay, a well-specified software system and the resulting high consistency in the basic offer-bid-acceptance-exchange framework make for a good fit with software's ruleishness.

2. *Online Markets Mitigate Transparency Problems Appropriately*

It is important to remember the senses in which markets are and are not transparent. Because markets themselves are not transparent with respect to the motivations of one's trading partners, it is irrelevant if an online market fails to provide this form of transparency. Indeed, many online markets thrive even in the presence of strong anonymity—all that matters is the deal itself.[104]

With respect to individuals' ability to know the terms of a deal, online markets, like any other markets, require a very high degree of transparency. In this context, transparency has typically been designed directly into these software systems; the software tries to be a neutral medium for presenting information about possible exchanges. The deployers of these systems have recognized that keeping their preferences out of the software is important in convincing people to use the systems. Typically the relevant rules are clearly explained, and information sufficient to verify most of their workings is made available.[105] This technique is especially reasonable when the market maker is not itself a transacting party but instead makes its money writing good software to make deals possible.

102. *See, e.g.*, Robert E. Scott, *The Death of Contract Law*, 54 U. TORONTO L.J. 369, 374-76 (2004).

103. *See* David Barboza, *On-Line Trade Fees Falling off the Screen*, N.Y. TIMES, Mar. 1, 1998, § 3 (Money & Business), at 4.

104. *See, e.g.*, F. Randall Farmer, KidTrade: A Design for an eBay-Resistant Virtual Economy (Oct. 19, 2004) (unpublished manuscript), *available at* http://www.fudco.com/habitat/archives/KidTrade.pdf (proposing an online market as part of a game for children that enforces complete anonymity in exchanges). Stock markets typically are close to anonymous in the offline world; the computerized systems that are replacing them are equally anonymous.

105. *See, e.g.*, eBay, *supra* note 95.

Some online markets have displayed hidden biases.[106] Amazon has attempted to price discriminate among its customers by tailoring special offers to their browsing patterns,[107] and there have been problems with insider access to mutual fund trading systems.[108] But such problems are not crippling in general. An online market is merely less effective in proportion to its lack of transparency, not something that will be useless if it falls short of perfect transparency.

3. *Online Markets Reduce Transaction Costs*

In most online markets, parties come together to make voluntary, binary exchanges. From this fact, two points follow. First, the parties to the deal are often placed in direct communication, and they can therefore directly negotiate terms left open by the software. For example, although eBay creates legally binding auction sales, it merely gives winning buyers and sellers each other's e-mail addresses and leaves them to arrange shipping terms.[109] Recognizing that software decisions cannot be ignored, eBay and many other online markets reduce the transaction costs of reversing incorrect decisions by enabling simple and direct negotiations by the transacting parties.

Second, even where the goods being exchanged have great value, the incentives to circumvent the software entirely can be kept low. Parties to market transactions are generally adversarial and will therefore police the system to make sure it is not in cahoots with their counterparties.[110] When the marketplace's cut is small, the fee savings enjoyed by parties who strike deals directly with each other generally do not justify the cost of negotiating such deals individually. The most notable circumventions of software restrictions target rules other than those enforced through software: for example, by suborning an institution with privileged access to the market.[111]

106. *See* Friedman & Nissenbaum, *supra* note 53, at 330-32.

107. *See* Paul Krugman, Op-Ed, *What Price Fairness?*, N.Y. TIMES, Oct. 4, 2000, at A35.

108. *See, e.g.*, Gretchen Morgenson, *Market Timing: A Longtime Practice Comes Under New Scrutiny*, N.Y. TIMES, Nov. 10, 2003, at C1.

109. *See* eBay, Unpaid Item Policy, http://pages.ebay.com/help/policies/unpaid-item.html (last visited Feb. 23, 2005).

110. These techniques are less effective against traditional techniques of transactional fraud. *See, e.g.*, Bob Sullivan, *Man Arrested in Huge eBay Fraud; Buyers Criticize Auction Site's Seller Verification Service*, MSNBC, June 12, 2003, http://msnbc.msn.com/id/3078461.

111. *See, e.g.*, Floyd Norris, *How To Beat the Market: Easy. Make Late Trades.*, N.Y. TIMES, Sept. 5, 2003, at C5.

4. *Online Markets Deal Appropriately with the Risk of Software Failures*

Several factors combine to make online markets robust in the face of software failures. First, however valuable the market itself may be, the spillover effects of a failure are not likely to include large consequential damages. Many online markets are careful to insulate the assets themselves from any failure of the software. If eBay's servers were to crash, none of the goods being traded there would suffer. Instead, after the interruption, trading would simply resume.[112]

Second, most online market makers are thoughtful about watching out for glitches, hacks, and crashes and have backup plans ready to respond to software failures. The results have generally been encouraging. While online markets have been subject to the occasional severe failure—some people blame computerized trading for the 1987 stock crash,[113] and e-commerce sites have been regular targets of hackers looking for large databases of credit card numbers[114]—there have been few reports of large-scale rigging or other manipulation of prices and exchanges. More problems have stemmed from the leakage of sensitive information than from deliberate market manipulation.[115]

5. *Summary*

Online markets attack problems that seem almost tailor-made for software-based resolution. Marketplaces benefit from the high ex ante certainty of the rules within which software wraps the contract formation process. The incentives for parties to transact around software's mediation can be kept low precisely because software can help drive the marginal cost of negotiation so low. Sensible policies by market makers have avoided thrusting such marketplaces into situations in which a failure of the software would hurt the underlying assets. The result of this sensitivity to software's limits has been a set of applications that are generally well accepted and uncontroversial, and often quite profitable.

112. *See, e.g.*, eBay, Outage Policy, http://pages.ebay.com/help/community/png-extn.html (last visited Feb. 12, 2005) ("Following a hard outage of two or more hours, eBay will extend the end times for all eligible listings by 24 hours.").

113. For a fuller discussion, see Richard A. Booth, *The Uncertain Case for Regulating Program Trading*, 1994 COLUM. BUS. L. REV. 1; and works cited therein.

114. *See, e.g.*, United States v. Ivanov, 175 F. Supp. 2d 367 (D. Conn. 2001).

115. *See* Bob Sullivan, *Credit Card Leaks Continue at Furious Pace; Security Firm Claims 120 Million Accounts Compromised This Year*, MSNBC, Sept. 24, 2004, http://www.msnbc.msn.com/id/6030057.

B. *Digital Rights Management Systems*

The Internet and other digital technologies have upended settled understandings of copyright. As a matter of policy, copyright is designed to give would-be authors and artists an economic incentive to be creative and to share the fruits of that creativity with the public.[116] Instrumentally, copyright produces this incentive by creating a legal chokepoint at the act of copying: Typically, anyone who wants to make a copy of a copyrighted work must obtain the copyright holder's permission.[117] Copying without authorization is deemed "infringement," and the infringer can be hit with a battery of civil, equitable, and criminal remedies.[118] Copyright is a classic form of regulation by law.[119]

The problem for this understanding of copyright in a digital age is that copying has become far, far easier than it used to be. Copyright infringement was once the province of those with printing presses; now anyone can be an infringer with the click of a mouse button. Existing notions of copyright enforcement by legal means no longer fit the technological landscape either. Unsurprisingly, the implications of this misfit are hotly contested. Some, most typically those with large portfolios of copyrights, argue for a ratcheting up of sanctions for infringement.[120] Others argue for redesigning copyright itself and finding some other technique for giving creators appropriate incentives to share their works with the public.[121] Both camps, however, are closely watching attempts to use DRM technology as a regulator instead of law, because such attempts could moot their debate or transform it beyond recognition.[122]

The goal of a DRM system is to allow an individual user to enjoy some piece of copyrighted content, thereby giving her a reason to pay for the system and the content, while preventing her from making a distinct copy of

116. *See, e.g.*, 1 MELVILLE B. NIMMER & DAVID NIMMER, NIMMER ON COPYRIGHT § 1.03[A] (2004).

117. 17 U.S.C. § 106 (2000).

118. *Id*. §§ 501-506.

119. *See* LAWRENCE LESSIG, FREE CULTURE: HOW BIG MEDIA USES TECHNOLOGY AND THE LAW TO LOCK DOWN CULTURE AND CONTROL CREATIVITY 121-22 (2004).

120. One such proposal was the Protecting Intellectual Rights Against Theft and Expropriation (PIRATE) Act of 2004, S. 2237, 108th Cong. (2004).

121. *See, e.g.*, WILLIAM W. FISHER III, PROMISES TO KEEP: TECHNOLOGY, LAW, AND THE FUTURE OF ENTERTAINMENT (2004); Raymond Shih Ray Ku, *The Creative Destruction of Copyright: Napster and the New Economics of Digital Technology*, 69 U. CHI. L. REV. 263 (2002); Ernest Miller & Joan Feigenbaum, *Taking the Copy Out of Copyright*, *in* SECURITY AND PRIVACY IN DIGITAL RIGHTS MANAGEMENT: ACM CCS-8 WORKSHOP DRM 2001, at 233 (Tomas Sander ed., 2002), *available at* http://www.cs.yale.edu/homes/jf/MF.pdf; Neil Weinstock Netanel, *Impose a Noncommercial Use Levy To Allow Free Peer-to-Peer File Sharing*, 17 HARV. J.L. & TECH. 1 (2003).

122. The analysis of DRM technology in this Section is strongly influenced by LESSIG, *supra* note 1, at 122-41, and works cited therein.

the content. She can enjoy the content within the software-regulated space created by the DRM system but cannot remove the content from that space.

To make this example more concrete, consider one of the most common types of DRM system in use today, an online music store like iTunes or Rhapsody. A user of these services pays with a credit card for individual songs, which are then downloaded onto her computer in an encrypted form. The encrypted form, by itself, is useless: In order to hear the songs, she must use a program provided by the online store that will decrypt the music file and play it through her computer speakers. However, the program will not give her direct access to the decrypted version. (If it did, she could send the decrypted version to a friend or use it in ways other than those intended by the DRM system's creator.)

These services differ in their details, but they have in common the idea that some uses of the song will be allowed by the software while others will not. DRM systems therefore employ software as a regulator to carve out a specific subset of allowed behavior from the larger space of the many things one could do with an unencrypted music file.

1. *DRM Systems Depend on Software's Ruleishness*

The restrictions that characterize DRM systems are fundamentally rule-like. They state precise conditions under which certain uses of the media will or will not be allowed. Although commentators disagree sharply on whether such ruleishness is a good way to control the use of media, they do not dispute that DRM systems use rules. Indeed, digital media themselves are creatures of software. An MP3 audio file "exists" only within a context defined by software tools. Digitized music and movies are already being made, copied, and distributed with software. Adding DRM technology to the mix does not require shifting behavior that currently takes place only offline into an online realm. Instead, it requires people to use one software application (a DRM-enabled one) in place of another application they would otherwise need to use in order to interact with the media files.[123] In this sense, DRM systems, like other digital systems for the enjoyment of media, depend on the predictability and cheap replicability of software rules.

On the positive side, to the extent that the allowable uses of media protected by DRM systems are made clear, consumers can develop precise expectations about what uses are and are not allowed.[124] Such precision

123. Thus, for example, iTunes will play either DRM-restricted files in the AAC file format or unrestricted files in the MP3 format. WinAmp plays only unrestricted MP3 files. There are formats even more "open" than MP3. The Ogg Vorbis compression technique is entirely patent free and is available under a license guaranteed to be royalty free.

124. It is true, of course, that these rules are precise in large part because they simply exclude an enormous number of uses that would be fully legal under existing copyright doctrine. *See* Julie

should be economically valuable, and one would expect it to lead to more efficient purchasing of media content. If this claim seems farfetched, consider the effects of another tactic being tested by recording companies: selling "CDs" designed not to play in computer CD drives.[125] These "CDs" also have a tendency not to work in certain CD players. Here, the buyer of music suffers from uncertainty about what uses she can and cannot make of the music. In comparison, a well-specified DRM system is much fairer to consumers, because they can develop reasonable expectations about it.

On the negative side, DRM systems reduce the social value created by fair use, and DRM technology is frequently opposed for this reason. According to this argument, fair use is and ought to be a standard and not a rule because only a standard will be attuned to individual equities. We should not expect a rule to capture all the subtleties of human creativity or all the possible uses we might wish to call fair.[126] The syllogism is simple: Rule-bound regulators are a poor choice for handling copyright; software is a rule-bound regulator; ergo software is a poor way to enforce copyright.

This debate is a rich one. It is also inseparable from one's normative views about the most appropriate level of copyright protection and whether copyright cases should be judged using rules or standards. There are certainly many who think that the vagaries of the fair use standards—especially the infamous "four factors"—count as defects and that we would be better off with a clearer rule, especially one under which more behavior was presumptively legal instead of being subject to expensive and uncertain litigation risks.[127] And as I have suggested above, if the fear is of insufficient complexity rather than insufficient clarity or insufficient ability to make post hoc course corrections, it is not clear that software's ruleishness is really the wrong sort of ruleishness for fair use. With these caveats, however, if one is committed to the idea that fair use embodies

E. Cohen, *A Right To Read Anonymously: A Closer Look at "Copyright Management" in Cyberspace*, 28 CONN. L. REV. 981, 1019-31 (1996). Cohen objects not to DRM technology's effectiveness or ineffectiveness as a regulator but instead to the substantive regulations DRM systems might effectively put in place.

125. Jefferson Graham, *Uncopyable CDs May Be Unplayable*, USA TODAY, Dec. 3, 2001, at 3D. Such "CDs" are often not real compact discs, because they have been altered so as not to comply with the Red Book standard that governs the technical specifications of CD audio. *See* Evan Hansen, *Dion Disc Could Bring PCs to a Standstill*, CNET NEWS.COM, Apr. 4, 2002, http://news.com.com/2100-1023-876055.html.

126. *See, e.g.*, Dan L. Burk, *Muddy Rules for Cyberspace*, 21 CARDOZO L. REV. 121, 140 (1999); Dan L. Burk & Julie E. Cohen, *Fair Use Infrastructure for Rights Management Systems*, 15 HARV. J.L. & TECH. 41, 54-70 (2001).

127. *See, e.g.*, LESSIG, *supra* note 119, at 187 ("But fair use in America simply means the right to hire a lawyer . . . ."). *See generally* David Nimmer, *"Fairest of Them All" and Other Fairy Tales of Fair Use*, LAW & CONTEMP. PROBS., Winter/Spring 2003, at 263, 279-84 (demonstrating statistically the malleability and unpredictability of the fair use standards).

important values in its use of a flexible standard,[128] then one has good reason to be skeptical of DRM software's ability to respect those values.

2. *DRM Systems' Lack of Transparency Raises Consumer Protection Concerns*

A concern about the inability of DRM systems to be fair judges of fair use might also be a form of skepticism about the transparency of DRM restrictions. This is very likely the nervousness felt by people who bump into surprising restrictions in DRM systems. One e-book version of the Constitution forbids printing;[129] some users have seen the music tracks they had purchased from iTunes stop working when they moved to another country;[130] DVD players prevent viewers from skipping the coming-attraction commercials on some DVDs, even on repeated viewing.[131] All of these are cases in which it might seem obvious that one should have every legal and moral right to do the deed in question—making it troublingly ambiguous whether the creator of the DRM system meant to prevent them.

In these examples, DRM technology helps to construct a system of copyright control in which decisional accountability is hard to find—but it does so without apparent malice. Foes of DRM technology fear that this blurring of accountability can also be the cover for more insidious doings, especially when a DRM system's design incorporates some measure of surveillance of the user.[132] The designer of a DRM system can incorporate hard-to-discern restrictions on its use. The newest version of iTunes, for example, has rules against repeatedly burning the same playlist to CD, but it is not clear exactly how different two playlists must be for iTunes to allow them both to be burned the maximum number of times.[133] Lurking in objections to such restrictions is a sense that your possessions—including your software—should be accountable to you and no one else.

Various proposals for an escape hatch in DRM systems bring together these strands of anxiety about the transparency of DRM technology as a regulator.[134] These escape hatches would allow users to do things normally

128. *See, e.g.*, Wendy J. Gordon, *Fair Use as Market Failure: A Structural and Economic Analysis of the* Betamax *Case and Its Predecessors*, 82 COLUM. L. REV. 1600, 1614-22 (1982) (arguing that fair use standards respond better to unpredictable market failures than would rules).

129. Lawrence Lessig, This Is the Constitution on DRM, http://www.lessig.org/blog/archives/001993.shtml (June 24, 2004 13:23 PST).

130. *See* Bob Tedeschi, *Out of the U.S. and Out of Luck To Download Music Legally*, N.Y. TIMES, July 28, 2003, at C3.

131. *See* Greg Sandoval, "Tarzan" *DVD Forces Viewers Through a Jungle of Previews*, CNET NEWS.COM, Mar. 2, 2000, http://news.com.com/2100-1017-237585.html.

132. *See, e.g.*, Julie E. Cohen, *DRM and Privacy*, 18 BERKELEY TECH. L.J. 575 (2003).

133. *See* Jason Schultz, Meet the New iTunes, Less Than the Old iTunes?, http://lawgeek.typepad.com/lawgeek/2004/04/meet_the_new_it.html (Apr. 29, 2004).

134. *See* Seth Schoen, *Give TCPA an Owner Override*, LINUX J., Dec. 2003, at 14, *available at* http://www.linuxjournal.com/node/7055/print.

forbidden by DRM systems without trying to hide the fact that the system had been overridden. The goal of such proposals is not to allow computer users simply to ignore all DRM restrictions, but rather to allow a form of human appeal. Such an escape hatch would be an explicit carve-out from regulation by software back to the world of regulation by law.

3. *DRM Systems Are Insensitive to User Transaction Costs*

Consider again the case of the music file that stops working when one moves to another country. The rule being applied here is supposed to restrict music purchasers to buying songs from a music store "in" their own country, as a way of tracking different national copyright and music licensing regimes. An individual who transports a personal music collection from one country to another is doing something that copyright law doesn't claim to reach and is happy to ignore. But the DRM software, unlike legal copyright enforcement, cannot be efficiently ignored by one on the right side of the law.

It is inherent to the purpose of a DRM system that it actively frustrate many possible transactions among its users. If I can find a way to move a protected media file out of a DRM system, I can continue to enjoy it using non-DRM-enabled tools, as well as share it freely with others. Further, the threshold for what counts as defeat is low. I do not need to cause the DRM software to stop running or to reprogram it to remove each specific limit. I just need to find (or create) a single hole in the system and export the file through that hole.[135] After that, I never need to engage with the DRM software again, because neither I nor anyone I give the file to needs to enter the DRM space to enjoy it. In sharp contrast to online markets, there are substantial incentives for users to band together to defeat DRM systems. Thus, such systems must adopt a more defensive, user-frustrating attitude.

4. *Software Failures Threaten DRM Systems*

Put another way, DRM systems suffer in a particularly acute way from the seriousness of software's failure modes. The list of famous failures of DRM systems is long. The system used to control the video content on DVDs was broken by a Norwegian teenager with a program simple enough to have been translated into song form.[136] The DRM system used to tie

135. *See, e.g.*, Crawford, *supra* note 5, at 618-21 (discussing a similar hole in technology mandated by a proposed FCC regulation).

136. *See* Universal City Studios v. Corley, 273 F.3d 429, 437-40 (2d Cir. 2001) (discussing the program); David S. Touretzky, Corley Gallery of CSS Descramblers, http://www-2.cs.cmu.edu/~dst/DeCSS/Gallery/index.html (last visited Feb. 12, 2004); *see also* Yochai Benkler, *Through the Looking Glass: Alice and the Constitutional Foundations of the Public Domain*, LAW & CONTEMP. PROBS., Winter/Spring 2003, at 173, 214-15 (describing the Gallery).

Adobe eBooks to a particular computer fell almost as easily.[137] Many DRM systems used to keep streaming Internet audio and video files from being recorded have been cracked,[138] as was every DRM technology employed in a famous challenge issued by the Secure Digital Music Initiative.[139] It is not clear that any DRM system has withstood a serious attempt to crack it.

In practice, these exploits have a common structure. Someone analyzes the DRM software. She then writes another program that takes advantage of a bug or design decision of the DRM system to defeat it, yielding a use of the content that the programmer of the system wished to prevent. Because not everyone is expert in reverse engineering, these programs can lead to mass defeat of DRM systems only if distributed to others. The original cracker, therefore, writes a program that can be used by others to defeat the DRM software.

In this sense, the use of DRM technology, while not effective as a system of regulation standing alone, does suggest a new possible chokepoint at the creation and distribution of such anti-DRM programs. Responding to the failure of software as a regulator, copyright holders have succeeded in enacting a legal system of regulation to take advantage of this chokepoint. The anticircumvention provisions of the Digital Millennium Copyright Act (DMCA) provide civil and criminal penalties for the creation, distribution, and use of "circumvention devices"—effectively, any technology (including software) that enables one to defeat a DRM system or avoid its restrictions.[140] The existence of the DMCA (and similar legislation in many other jurisdictions) is an open admission that software has failure modes sufficiently severe that regulation by software alone cannot be trusted. In that sense, the DMCA is a targeted response to the admitted weaknesses of software. The effectiveness of DRM software as a regulator is therefore dependent on the legal effectiveness of the DMCA.

Many authors argue, for reasons related to the conception of software in this Note, that the DMCA's harms outweigh the good it does in propping up DRM technology.[141] The DMCA creates a category of per se illegal software by outlawing programs that do certain things. But in so doing the DMCA aligns itself squarely against software's plasticity. In making it illegal to create certain kinds of software, it tries to prevent people from

137. *See* United States v. Elcom Ltd., 203 F. Supp. 2d 1111, 1118-19 (N.D. Cal. 2002).

138. *See, e.g.*, RealNetworks v. Streambox, No. C99-2070P, 2000 U.S. Dist. LEXIS 1889 (W.D. Wash. Jan. 18, 2000).

139. *See* Scott A. Craver et al., *Reading Between the Lines: Lessons from the SDMI Challenge*, *in* PROCEEDINGS OF THE 10TH USENIX SECURITY SYMPOSIUM 353 (2001).

140. 17 U.S.C. §§ 1201-1205 (2000); *see also* ELEC. FRONTIER FOUND., UNINTENDED CONSEQUENCES: FIVE YEARS UNDER THE DMCA (3d ed. 2003), http://www.eff.org/IP/DMCA/unintended_consequences.php; Lawrence Lessig, *Law Regulating Code Regulating Law*, 35 LOY. U. CHI. L.J. 1 (2003).

141. *See, e.g.*, Solum & Chung, *supra* note 6, at 944 (criticizing the "layer-crossing" feature of the DMCA).

taking full advantage of one of software's greatest virtues: that software can do almost anything one can imagine.

5. *Summary*

The regulation-by-software framework reveals that DRM technology raises serious concerns that online markets do not. As software systems go, DRM systems encourage their own circumvention and hold up poorly in response to attack. Their feasibility therefore depends on extensive external investment, both social and legal. The inflexibility of DRM systems also regularly frustrates users' desires to use protected content creatively, raising transaction costs. DRM systems' potential lack of transparency may be troubling, but well-designed systems mitigate this problem with good disclosure of the underlying rules and good responsiveness to complaints. And whether DRM systems' ruleishness is good or bad depends on one's other policy commitments. All in all, DRM technology as a form of regulation by software deserves a yellow warning sign: Proceed with caution, danger ahead.

That said, fairness to DRM systems requires that they be evaluated in the context of other proposed solutions to the computer copyright crisis. Leaving aside serious reform of copyright law itself, most such solutions depend just as strongly on massive software systems as DRM-based solutions do. They are vulnerable to exactly the same criticisms.

One commonly proposed solution, for example, consists of a collective license. In exchange for paying a copyright levy, consumers would be allowed to copy content freely without fear of suit.[142] This levy would then be divided among the creators of such content. Although both the collection and the distribution halves of this proposal come in many different varieties, no one has yet told a story in which neither half involves a large-scale software system. Where such a system can be identified, the vices of software can be exploited.

Digital media are at least the right sort of thing to be regulated with software, because they are themselves creatures of software. This theme also caps the effectiveness of any purely software-based system of regulating digital media. The plasticity of software makes it all too possible to write more software that handles the regulated media in a way the system regulating them cannot stop. Software can do some work in controlling the

142. *See, e.g.*, ELEC. FRONTIER FOUND., A BETTER WAY FORWARD: VOLUNTARY COLLECTIVE LICENSING OF MUSIC FILE SHARING (2004), http://www.eff.org/share/collective_lic_wp.php; FISHER, *supra* note 121; Netanel, *supra* note 121; Lionel S. Sobel, *DRM as an Enabler of Business Models: ISPs as Digital Retailers*, 18 BERKELEY TECH. L.J. 667, 680-84 (2003).

uses of digital media, but not all the work. The most appropriate use of software here will be one that respects these limits.

CONCLUSION

We are all regulated by software now. It has become possible to imagine that the most basic aspects of democracy, society, and even life itself might be regulated by software. The federal government has tried to regulate privacy,[143] advertising,[144] and pornography[145] by software. In an age when software promises to do anything, perhaps anything can be regulated by software.

But regulation by software is never required. The very properties of regulation by software that make it attractive for some areas make it positively dangerous for others. Appreciating the similarities common to all applications of software makes it easier to think critically about particular uses. Each case in which we have used software tells us a little more about what to expect if we turn to software to resolve the next one. The choice for software may sometimes be justified and sometimes not, but there is no excuse for making that choice blindly.

143. *See* Health Insurance Portability and Accountability Act of 1996, Pub L. No. 104-191, 1996 U.S.C.C.A.N. (110 Stat.) 1936 (codified in scattered sections of 18, 26, 29, and 42 U.S.C.).

144. *See* Controlling the Assault of Non-Solicited Pornography and Marketing (CAN-SPAM) Act of 2003, Pub. L. No. 108-187, 2003 U.S.C.C.A.N. (117 Stat.) 2699 (to be codified in scattered sections of 15 and 18 U.S.C.).

145. *See, e.g.*, Child Online Protection Act § 1403, 47 U.S.C. § 231 (2000), *invalidated by* Ashcroft v. ACLU, 124 S. Ct. 2783 (2004); Communications Decency Act of 1996 § 502, 47 U.S.C. § 223, *invalidated by* Reno v. ACLU, 521 U.S. 844 (1997).