27 patterns to keep your users coming back for more

Irresistible Apps

Motivational Design Patterns for Apps, Games, and Web-based Communities

Chris Lewis

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Chapter 1

Introduction to Motivational Design

This introductory chapter presents a broad overview of what this book is all about. Different readers will want different things out of this book, and it's this chapter's job to get you up to speed quickly, so you can begin to pick and choose how you navigate through the other chapters.

In this chapter, you'll learn

- What motivational design is, and where you can go looking for examples in your own life
- What motivational design patterns are
- How to read this book, teaching you how to get the information you need quickly

Motivational Design

We live in an era that is quite unlike any other we have seen before. The products that we build today as software designers can have scale, reach, and velocity that have never been matched. One student in a dorm room hacking away at a web site can create a company called *Facebook* and grow it to an audience of 1.11 billion monthly active users in just nine years. That's one active user for every seven on the planet. If speed is what you are looking for, Zynga's *CityVille* social game grew to 100 million active users in just 41 days. To put that number in perspective, only 11 countries have populations greater than 100 million people! *CityVille* was not so much a city as it was a global superpower.

What is it about *certain* pieces of software that causes them to be so popular? What are the secrets of building *irresistible software* that *motivates* users to return again and again? This book pulls back the curtain to reveal these hidden design techniques, called *motivational design patterns*. You'll get a

look at the psychology of your users, so that you can learn what drives them, then gain access to a library of patterns that you can use in your own software, appealing to your users' core motivational needs. This is what separates functional software from *irresistible* software.

The Irresistible Smartphone

For a quick example of irresistible software, take a look at your smartphone's application list. There are probably applications that offer the means to connect with others, such as e-mail or *Facebook*. Likely, there are some news applications, so that you can make sure no event has passed you by. Maybe there are some games that you poke and prod at every once in a while. None of these applications seems terribly *important*—and, hard as it is to contemplate, we did live our lives quite happily before the iPhone came out—but we miss our smartphones when we accidentally leave them at home.

The draw of the smartphone is undeniable. A quick look around airports, supermarket queues, and coffee shops will show a number of people all doing the same thing—mooching around on their phones as time passes. As Ian Bogost puts it, "It's not abnormal. It's just what people do. Like smoking in 1965, it's just life." Smartphones are wonderfully *immediate*. When we need to reach out to someone, they're only a couple of taps away. When we feel curious, *Reddit* always has something new. When we want to compete, *Hero Academy* lets us fight our friends.

Smartphones provide us with easy access to many things we fundamentally desire. By providing that access, we find their siren song hard to resist, constantly calling out to us. We feel elated when we download an app that allows us to do something we couldn't do before or show us something we previously hadn't seen. We get equally frustrated when an application promises something that it then fails to deliver, and we are equally quick to delete apps as we are to download new ones. If you want to find more examples of irresistible software when reading this book, just reach into your pocket and pull out your smartphone. The applications that have survived your ruthless culling are the ones that are *truly* irresistible.

The Zero-Sum Game

The reason why more and more developers are trying to design irresistible software is to increase user retention. Users have come to expect immediate satisfaction without large upfront costs, and so have begun to abandon the old way of one-time purchases of software in boxes. Successful companies like Google, Netflix, and Supercell look to advertising, subscription models, or in-app purchases. These revenue models hinge on keeping users engaged and happy, so they continue returning to click advertisements, renew their subscriptions, or purchase more in-app goods. Even the venerable *Microsoft Office*, one of the defining products of the boxed software era, now allows users to subscribe to the software monthly, streaming the suite of applications to the user's hard drive over the Internet. User retention is what pays the bills. However, when all applications are built to be available and immediate, users can (and do) leave one application for another. In the vicious Software Market Thunderdome that the Internet has created, each application needs to continually prove itself against an onslaught of hungry challengers.

The amount of time users can dedicate to applications isn't flexible: they each only have so many minutes in the day to spend fiddling with smartphones or noodling around the Internet (unless, of course, they do not mind being fired or are unmoved by the threat of divorce proceedings). The way users share time with software resembles a concept that economists call a *zero-sum game*. A classic zero-sum game is poker: when someone has won some money, someone else has lost it. Any time that users dedicate to one piece of software will be at the expense of another, so software has to have a competitive advantage to take a bigger cut of that time pie. Irresistible software is not just an interesting offshoot of software design. It's at the core of software that's designed in this zero-sum environment. The best software designers will be those who can compete for the hearts and minds of users. Motivational design patterns provide an easy-to-use toolkit to create irresistible designs, providing the competitive advantage a designer needs to attract a loyal user base. And loyal user bases mean more advertising clicks, subscription renewals, and in-app purchases. The maxim of time is money has never been truer for the software industry.

Motivational Design Patterns

Many readers may well have experienced design patterns at one time or another. Later on, in Chapter 3, I'll delve into design patterns in more detail, but it's worth describing here why they're useful and what you'll see later.

Design patterns allow us to express the commonalities between different designs. No design is an island, and designers borrow and evolve ideas from one another. Design patterns provide a means of identifying these common ideas, allowing us to name and describe them and then use them in our own designs. Motivational design patterns describe common aspects of software that fulfill basic needs within all of us. We are *intrinsically motivated* to seek out things that we find interesting. Motivational design patterns provide interesting and useful mechanisms that users will come back to again and again. The collection of patterns in this book includes patterns that address some of the most important things in our lives (being able to search and find one beloved long-lost photo), right down to things that appear trivial at first glance (such as the praise we get from the sound of Mario grabbing a coin). The library of patterns provides a language to describe the similar designs that exist across different pieces of software, and theories from motivational psychology, behavioral psychology, and behavioral economics are used to describe their motivational power. These theories also help explain whether certain pattern usages are effective and what can be done to improve poor uses of a pattern.

Psychology theories also help us to talk about *dark patterns*, the patterns that cross the line from being *motivational* to *manipulative*, and these patterns to avoid using can also be found in this book.

How to Read This Book

You will get the most out of this book by reading Chapter 2 next. This chapter outlines much of the psychology that the patterns rely on, and patterns will often refer back to ideas presented in that chapter. If you need a primer on what patterns are and why they're useful, you can read on to Chapter 3. You should then feel free to jump around the pattern library as you see fit, finding patterns that seem suitable for the project you're working on.

I have used screenshots in this book to better illustrate how a design works. Where including a screenshot was not possible, I created wireframe mockups that closely resemble the software instead. Any differences between the mockups and the software itself are minor and won't affect the point being made.

To help you navigate, here's a short description of each pattern in the library.

Chapter 4: Gameful Patterns

These patterns have the qualities of gaming and focus on quick feedback loops.

Collection: Collections let users build up sets of virtual items. The joy of collecting and owning these goods are directly used in many games, most notably driving games and, of course, *Pokémon.*

Specialization—Badge: An indicator of reaching a certain goal. Most famously used by Microsoft in the Xbox Achievements system, badges are a means for users to recognize their progress and prowess.

Growth: Ownership of something that grows over time. Growth surprises and delights users, providing them with something uniquely their own.

Increased Responsibility: This pattern lets trusted users perform more influential actions. Users feel great that they're being recognized for their service, while taking on more work to curate the application that they're involved with.

Leaderboard: Leaderboards place users in ranked lists of others, providing them with the means of comparing their abilities against others. Leaderboards encourage the competition in users, driving them to return to the application to try and get a higher position.

Score: Points awarded in response to actions. While a simple method of providing users with feedback about their position, Score is used and abused in equal measure. The simplicity masks hidden dangers that designers need to be careful of.

Chapter 5: Social Patterns

These patterns help users fulfill their *Social Contact* needs.

Activity Stream: A series of broadcasts grouped together into a single list. Activity streams provide a place for users to find out what's going on in an application. They can change an application from looking like a ghost town to that of a bustling metropolis.

Broadcast: A means for a user to share information with others, the broadcast pattern is the linchpin of all other social patterns. When users send out status updates, photos, and chat messages to one another, they're broadcasting that information.

Specialization—Social Feedback: A means for a user to send feedback about a broadcast. Be it a "Like," a comment, or a reshare, all users who post broadcasts are secretly (and not so secretly) hoping for some social feedback.

Contact List: A list of contacts that allows the user to directly interact with an individual on the list. Contact lists are how users direct broadcasts to specific people.

Identifiable Community: An area where a group of like-minded individuals can come together and interact with one another. These forums allow the exchange of information, jokes, and, ultimately, friendship, becoming a daily must-visit application for users who are fully invested.

Specialization—Meta-Area: A place for community members to guide the product and formed of one or more identifiable communities. Meta-areas get users involved in product development, growing their investment in the product while they provide helpful feedback to focus future development.

Identity Shaping: When users express some facet about their avatar online, they are shaping their identity. From simply adding a nickname all the way to curating blogs entirely about Justin Bieber, users curate identities that they share with others, either to self-affirm or to experiment with other personalities that they cannot use in real life.

Item Sharing: Item sharing is the exchange of virtual goods between users. Sometimes they're gifts and sometimes they're trades, but this pattern can help users grow their Collections at the same time as they reinforce their social bonds.

Chapter 6: Interface Patterns

Interface patterns describe how applications communicate to the user through the interface.

Notifications: Notifications alert users that there's been some change of state in the application. Sometimes these alerts prompt the user to take action and sometimes they just pique curiosity, but they're always difficult to ignore.

Praise: This pattern rewards users for performing actions. Users feel good, and they learn that they can come back again and again for another hearty pat on the back. Who doesn't want constant approval?

Predictable Results: Actions taken should have predictable results. If users are unsure about what effect an action might take, they become nervous. Those nerves not only prevent the user from performing that particular action but make them nervous about taking other actions too.

State Preservation: Applications that can be exited at any time should preserve their state, so users feel confident that they can drop in and out of the application at will and never fear that they will lose progress.

Undo: Actions that can be reverted can be explored. Exploration lets users unleash their creativity, safe in the knowledge they can undo something they don't like.

Chapter 7: Information Patterns

These patterns guide users through content, often satisfying their *curiosity* needs.

Customization: Users can customize their virtual space, making it their own. As Gabe Zichermann puts it, "Customization is commitment."

Specialization—Filters: Content can be highlighted or hidden, allowing users to customize the way they view information.

Intriguing Branch: Interesting content is linked, letting users explore intriguing branches. Anyone who only needed to look up a quick piece of information about the Ford Mustang on Wikipedia only to escape two hours later having somehow reached an article about the native flora and fauna of the Indonesian archipelago knows just how powerful intriguing branches can be.

Organization of Information: Information that can be organized for later retrieval makes users feel safe that their data is always going to be easily found.

Personalization: Systems that personalize themselves to the perceived needs of the user are able to surface information and functionality without the user having to look. *Amazon's* recommendation system is the most famous use of this pattern.

Reporting: Content that users deem unacceptable can be reported. This lets users feel like they're honorable citizens of your application, while they help ease the moderation burden from the developers.

Search: When users can search for content, they feel confident that their data will never be truly lost.

Task Queue: Task queues are the shopping lists of applications. They tell users what they can do next, always providing them with new, interesting tasks to perform.

Chapter 9: Temporal Dark Patterns

Temporal dark patterns occur when users are unable to correctly estimate how much time they will interact with an application. This can occur when the application requests *too much* time from the user or when the application offers *too little*.

Grind: When users repeat a skill-less task in order to progress, they're grinding. Grinding gives the impression that users are doing something worthwhile when, in fact, they are simply wasting time.

Hellbroadcast: Filtering a user's broadcasts without consent results in a hellbroadcast. When a user is put in this special circle of application hell, her time is wasted creating broadcasts that no one can see.

Interaction by Demand: The pattern forces users to engage with the application on its own schedule, regardless of whether the user can actually afford to dedicate the time to do so. Users who feel nagged are far more likely to delete an application than they are to heed its calls.

Chapter 10: Monetary Dark Patterns

These patterns cause users to either lose track of or regret spending money, creating a short-term gain for the company but resulting in long-term loss of motivation in their audience.

Currency Confusion: Substitution of money for an arbitrary currency confuses users as to what the exchange rate for purchases actually is. Often this means they end up spending more money than they intended.

Monetized Rivalries: When users are pitted against one another, their competitiveness can be exploited by offering paid-for upgrades. In the heat of the moment, users may well purchase something they later regret.

Pay to Skip: Users can pay money to skip onerous issues, usually grinds that have been arbitrarily added in the hope that users will part with their money to avoid them.

Chapter 11: Social Capital Dark Patterns

Social capital dark patterns exploit a user's social network, putting her friendships at risk.

Impersonation: Creating broadcasts that appear to be from the user but are in fact generated by the application. If the posts result in a negative reaction, others' view of the user deteriorates. How many times have you seen the spam from a social game on a *Facebook* wall and lowered your opinion of the user who supposedly posted it?

Social Pyramid Schemes: A requirement for other people to be brought in to the application before it is interesting. Applications should always be interesting, without having to constantly hound your significant other, parents, and pets to create new accounts.

Conclusion

By now, you should have a good idea of what motivational design is and how motivational design patterns provide a toolbox for you to apply motivational ideas into your own software. You've also had a sneak peek into the design patterns that the library contains. In the next chapter, I'll go over some of the psychology theories I'll be referring to in the rest of the book. If you're feeling confident enough to dive right in, feel free to look up some of the patterns in the book that interest you. You can always come back later!

And so, let us put on our pop psychologist hats and venture off into Chapter 2.

Chapter 2

Psychology of Motivation

This chapter provides the core psychology grounding that you'll need in order to truly comprehend *how* motivational design patterns work. Too much of what you may have read in other sources simply presents a design "trick," without justifying if or how it has the desired effect on users. It's my goal to take you deeper into the science of psychology, not just to show you how the existing motivational design patterns work, but to give you the understanding you'll need to come up with your own patterns.

In this chapter, you'll learn

- About the danger of "cargo cult design"
- What "behavioral psychology" and "Skinner boxes" are, as well as the weakness of focusing only on these ideas
- What intrinsic motivation is, how to harness it, and the multifaceted model of intrinsic motivation that each pattern is built upon

Cargo Cult Design

Before we can begin to discuss motivational design patterns, we must first understand exactly what motivation is. This is often the missing link in many conversations surrounding product development, which often seem to assume that "if you build it, they will come." But is this true? What's this assumption based upon? Prior success of the company? Prior successes from other companies? Without really *understanding* the reasons behind why some software grows exponentially and some software fails in a few short days, the difference between future success and failure is as much attributable to luck as it is to design. That said, it's understandable why we, as software creators, don't want to delve into the minds of our intended audience. That's a whole other field of training. To make matters worse, psychology is a wide and varied space, full of argument and contradiction, without the certainties of input/output that we're used to as computer engineers. Humans are abstract, diverse, and irrational. It's reasonable to ask why we should bother looping in psychology at all.

The problem with a try and see approach is that it leads to "cargo cult" design. The cargo cult term comes from cults in small, preindustrial tribes in the Pacific. These tribes were exposed to cargo coming from Western societies, but eventually, the cargo would cease to arrive as the Westerners left. As the tribes didn't understand where the cargo came from, they turned to rituals to try to re-create the *conditions* at which the cargo arrived. In the case of World War II, they built faux-airstrips and radio equipment!

We see this exact same behavior when we hear phrases such as "we need to make it more social" or "let's add a gamification layer." The designers are trying to re-create the conditions that provided success for others, without understanding the core psychological foundations of what drove that success in the first place. We need to understand why people are motivated to engage with a software product, so that we can make the right choices about what to add, what to leave out, and be able to identify what is missing. Otherwise, we remain in a cargo cult state, attempting to replicate only what we have seen, without any knowledge that what we are doing will result in the right outcome.

To gain the required understanding, you'll need to go on a whirlwind tour through three key subjects: behavioral psychology, intrinsic motivation theory, and behavioral economics. While this book is perhaps somewhat cavalier at mixing and matching these fields, it's important to note that they are distinct, with their own theories and experimental results. To minimize any negative impact from conflating these fields, we'll only consider theories that have empirical data to support them.[1] What we are interested in as software designers is not so much the theory of mind but the ability to make an informed guess as to how a user may respond to a certain pattern. This pragmatic approach lets us get at the core ideas of how we might design software, without getting lost in theoretical frameworks that could contradict one another.

Behavioral Psychology

Behavioral psychology is a perspective that organism behaviors occur as responses to stimuli. Certain *stimuli* (inputs) are introduced to the body, and certain *responses* (outputs) occur. If you are poked with a stick as the stimulus, you'll probably yelp as a response. This is a fundamentally *extrinsic* view of our motivation: we modify our behaviors in reaction to our environment.

The most famous of the behavioral psychologists was B. F. Skinner. He held a particularly functional view of behaviorism: what people think deep down is so inherently imprecise and fuzzy that science should just focus on observable behavior that's easily tested and quantified. Our behaviors are shaped by the environmental stimuli around us, so Skinner's experiments focused on tinkering with the environment of test subjects and observing how their behavior changed.

[1]Some readers may wonder why research from game designers does not receive attention in this section. The goal of this chapter is to introduce psychological concepts that help to explain intrinsic motivation at a *fundamental* level, so that such knowledge can be generalized across all kinds of software, without gameful contexts being suggested as required to create intrinsically motivating software. This psychology research, then, helps to *explain* the insights that game designers have shared. Literature from game designers will be used throughout this book, and their exclusion here should not be taken to imply that their work is not useful.

To this end, he's most well-known for the "Operant Conditioning Chamber," which is now often referred to as a "Skinner box." Using it, he would study *operant conditioning*, looking at how reinforcement or punishment led to the increase or decrease of certain voluntary behaviors. Operant conditioning is all about learning how to do well in a certain environment, performing behaviors that maximize the things we like (dating a particularly cute guy/girl or making money) and minimizes the things we don't (getting dumped or getting fired).

When users are in computational environments, they behave just as they would in any other environment: they try to maximize the good outcomes and minimize the bad. Good designers help users along, creating environments that help the user find those good results ("I found that long-lost picture of Auntie Anne!") while avoiding the bad ones ("I deleted that long-lost picture of Auntie Anne by mistake!"). One way of doing this is to create an environment that's familiar to others, so users don't have to explore an interface (and possibly hit a landmine along the way). It's for this reason that we see *conventions* everywhere in user interfaces, even across different operating systems. Think of the save icon. It's a floppy disk. When's the last time you saw a floppy disk in real life? If you were born after 1995, the answer is probably never. And yet designers cling to the floppy disk icon because users have been conditioned to know that clicking the floppy disk icon will result in their document being saved.

Skinner constructed the Skinner box to perform tests on rats and pigeons, to see how they would respond to certain stimuli. The most well-known use is a lever that a rat can pull to get food, leading the rats to pull the lever more often to get food. The food is called a reinforcer: it's an extrinsic motivator that increases behavior. Skinner then started to play with when food was produced. Sometimes the food would come out on every pull. Sometimes it would come out on every tenth pull. Sometimes it would come out at an average of one out of ten pulls. Sometimes it would come out only after a certain amount of time had passed. By affecting the environment that he controlled, Skinner found he could condition the *voluntary actions* of the rats and pigeons. Controlling voluntary actions is what we are concerned with when we speak about irresistible apps: we want the user to voluntarily interact with our application.

The exact setup for how and when rewards are offered is known as a schedule, and one schedule in particular will be referenced later on: the variable ratio schedule. The various reward schedules are

> **Fixed Ratio:** Delivers the reinforcement after every *n*th response. A coffee card that gives a free coffee after nine cups would fit under this heading.
>
> **Variable Ratio:** Delivers the reinforcement *on average*, after *n* responses. This is the classic "slot machine" schedule used by one-armed bandits the world over.
>
> **Fixed Interval:** Reinforcement is delivered after *n* period of time. An automatic coffeemaker runs on a fixed interval, providing a (hopefully) warm cup of coffee after a certain amount of time.
>
> **Variable Interval:** Reinforcement is provided at an average interval of *n* time. Fishing is a good example of this. A fisherman might catch a fish after just one minute, or he might have to wait an hour to get a bite.

The variable ratio schedule creates the most response over time. If one wants to create an addictive experience, the variable ratio is the one to choose. Unsurprisingly, the science around this schedule has been honed to a fine level of specificity by the casino industry. And this is the key behavioral

psychology finding we're interested in. We'll see the variable ratio come up multiple times throughout this book. If you're thinking you need a mechanism to bring a user back to your application, think variable ratio.

Psychologists generally agree, however, that we can go deeper into the psyche than Skinner did. Not everything has to be built around reinforcers and rewards, but this is what gamification and its corresponding touchstone book, *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps,* by Gabe Zichermann and Christopher Cunningham (O'Reilly, 2011), uses exclusively. Many of the patterns listed in that book revolve around rewards—such as score, leaderboards, badges—and present them as motivating in and of themselves. But these are the only the tools with which things can be built; they don't tell us why some things work and why some things don't. Jon Radoff expands on this issue in his book *Game On*:

> *The problem with gamification isn't the term, or its objectives, but how it is applied… It's the behaviorist approach to games that channels inquiry away from the harder problems of immersion, cooperation and competition that is so important to creating successful game experiences. Behaviorism was popular in psychology because it seemed to offer some easy answers—some of which do work (such as certain forms of conditioning) yet which is built on an erroneously reductive premise that ultimately failed to be supported empirically.*

When you sit down to design an irresistible app, you have to be on your guard for easy answers such as a behavioral approach. It misses the depth of experience that you must create in order to have the long-lasting, *meaningful* attachment that products need in order to compete for users' time. Gamification erroneously uses those easy answers from behaviorism and then, in turn, presents them as the easy answers for how to increase user retention.

An easy way to remember this point comes from Jesse Schell, who used the thought experiment of "chocofication." The story goes like this: chocolate tastes great. Chocolate with peanut butter tastes even better. But you can't conclude that chocolate makes *everything* better. Adding chocolate to hot dogs is a disaster when what you need is mustard. Chocolate is no easy cooking answer, just as gamification is no easy design answer. Aiming merely to gamify your app might well be adding the proverbial chocolate on your software hot dog.

Intrinsic Motivation Theories

Intrinsic motivation comes from within, whereas extrinsic motivation comes from without. It's what motivates us to do things only for the joy of doing them, and we do them even if there are no environmental reasons to do so. It's what pulls us to play another hour of *Halo* rather than write essays for a class, even though we might be paying large amounts of money to attend that class.

When we think of trigger words such as *interesting* or *fun*, we're thinking of intrinsic motivation. When we engage in a task even when our environment encourages us not to (such as surfing *Reddit* during work hours, at the risk of losing our job), we're engaging in an intrinsically motivating task. This is what separates irresistible apps from anything else. Users open them because they want to. It sounds simple, but think of the number of apps you've tried on the Web or your phone once and never returned to. Creating an application that is intrinsically motivating is the hard part.

In this section, several different researchers on intrinsic motivation will be presented. Malone and Deci and Ryan are largely complementary researchers, whereas Reiss has a separate view of intrinsic motivation. The researchers are presented in chronological order: Malone, Deci and Ryan, and, finally, Reiss.

The Importance of Learning: Malone

Thomas Malone was a graduate student at Stanford University when he first began formulating his ideas about intrinsic motivation. He was certainly not the first person to work on intrinsic motivation, but he was the first to look at the issue of intrinsic motivation and software. In his scientific white paper "Toward a Theory of Intrinsically Motivating Instruction," he recognizes playing video games as an intrinsically motivating activity for some people (of course, games also contain extrinsic motivations, such as scores and achievements) and tries to pick apart what makes games captivating, using a version of *Breakout* that he created. What's particularly interesting is that he identifies a number of things he thinks make an educational video game motivating, such as that it provides goals, increases the user's self-esteem, and offers choices. Although Malone only focused on educational games, his findings seem applicable to all environments where we require motivation. When we think about learning, we imagine classrooms and lecture theaters. In fact, learning seems to be a core part of any motivational environment, be it classroom, workplace, or home.

Raph Koster's *A Theory of Fun for Game Design* (Paraglyph, 2003) describes this in the context of video games: "With games, learning is the drug…When a game stops teaching us, we feel bored." He even expands this to situations where it isn't clear that we are learning: "When you feel a piece of music is repetitive or derivative, it grows boring because it presents no cognitive challenge… [the brain] craves new *data*." Motivational software often contains a learning element with which users are surprised, perhaps even delighted, to learn something new, either in the interface (such as finding some cool functionality they didn't know about) or in the data the interface presents (such as seeing the latest baby updates from a friend on *Facebook*).

Malone's findings show that offering new things to learn is one way we can create software that's intrinsically motivating. We'll now look at the work of Edward Deci and Richard Ryan, who offer another framework we can use when trying to create intrinsically motivating designs. They believe intrinsic motivation comes down to just three core ideas: autonomy, competence, and relatedness.

Autonomy, Competence, and Relatedness: Deci and Ryan

Edward Deci and Richard Ryan were two professors based at the University of Rochester. Together they worked on a concept known as Self-Determination Theory. The joy of Self-Determination Theory is that it succinctly describes things humans find intrinsically motivating, relying on just three core ideas: autonomy, competence, and relatedness. From that, they also worked on an idea called Cognitive Evaluation Theory, which studied ways in which feelings of intrinsic motivation could be hampered.

Self-Determination Theory

Self-Determination Theory (SDT) is a theory of motivation first introduced by Deci and Ryan in 1985, but it wasn't until Daniel Pink released his book *Drive* in 2011 that the idea became mainstream. SDT defines just three core tenets that a task must have in order to be intrinsically motivating.

> **Autonomy:** The ability to make choices as you see fit; being the perceived origin of your behavior. This does not necessarily mean that you are independent (not relying on the help of others) or that a choice is not forced on you by someone else (you have autonomy if you feel the decision is correct). Autonomy also does not necessarily imply having a wealth of options, as long as the options available present the path you wish to follow. For example, first-person shooters don't offer many options. *Half-Life* doesn't offer you the chance to sit down and have a roundtable discussion about whether the aliens should end their invasion. However, it does offer the chance to dispatch them with a variety of weaponry, and this is the choice that the audience of *Half-Life w*ants anyway. This conclusion means it's important for software designers to identify just *what* their audience really wants to do. Too few options, and the users don't feel empowered to do what they want. Too many options, and the software becomes too complex to operate, again disempowering users from taking the actions they wish.
>
> **Competence:** That the task at hand is something by which you feel challenged but is likely achievable. The challenge should be "optimal for [your] capacity," and allow you to grow your abilities and gain mastery of situations.
>
> **Relatedness:** That the task creates a feeling of connectedness to others—caring for them and them caring for you. Pink expands this notion slightly by renaming it *Purpose*. The task creates a meaningful change that leads to something bigger than just ourselves. Connecting with others is a purposeful task, so relatedness is a subset of purpose. Importantly, other research described in Scott Rigby and Richard Ryan's *Glued to Games* (Praeger, 2011) found that relatedness can come not only from real humans but from virtual characters as well. Saving the village in *Skyrim* can feel just as meaningful as helping out with your local school's bake sale.

This, in a nutshell, is the entirety of SDT. It's intuitively believable, and we can imagine times in our lives, particularly in the world of work, where we felt that we had such things and were really motivated. We could do what needed to be done, the work was interesting and challenging, and the results provided something that felt important. But many of us have also had that job where our autonomy was thwarted at every turn, the challenge was not there, and there was no purpose to what we were doing.[2]

[2]As a teenager, I had a particularly miserable summer job, tasked with using *Microsoft Word* to write out invoices from a database. "Why doesn't the database just generate them?" I asked. "Because it doesn't, and why would you even think that it could?" was the response. "I could make the database do it in about three weeks, if you give me the right software," I answered. I was denied and told to get back to work. The knowledge that my job was essentially a cog that any computer could do was crushing. My autonomy to actually do the job better was gone; there was zero challenge in moving data from one program to another; and the task didn't exactly help further the goals of humanity.

One other benefit of SDT is that its general breadth covers a wide spectrum of applications.[3] Unfortunately, this also makes it more difficult to apply with any granularity.

Cognitive Evaluation Theory

Cognitive Evaluation Theory (CET) is a subset of SDT that focuses on how extrinsic rewards affect intrinsic motivation, focusing exclusively on the autonomy and competence aspects of SDT. A reward doesn't just have to be a trinket or food, it can be something as simple as being verbally praised. CET posits that when a feedback event occurs that we perceive as being *informational* of our mastery of something, we use this to satisfy our intrinsic need for competency. Without information on how we are doing, we have no basis for understanding whether we're getting better at it. However, if the event is seen as *controlling* us, we lose our feelings of autonomy, and our intrinsic motivation drops.

This theory strikes at the heart of an ongoing and unresolved tension in the motivational psychology community as to whether extrinsic rewards undermine intrinsic motivation. The classic supporting example comes from an experiment with children who enjoyed to draw.[4] The children were split into three groups. The children in one group were told they would get a shiny gold star with a red ribbon if they drew a picture. The second group was given the star for drawing the picture but was not told ahead of time that members would receive one. The third group was not made aware of the star nor given one. The results showed that the group that had been told about the star beforehand drew fewer pictures independently afterward. Members of the other two groups showed no change. The theory is that the first group had succumbed to the overjustification effect: the children became focused on the extrinsic reward and rationalized to themselves that they had drawn the picture for the reward, not for the joy of drawing the picture. They had overjustified the point of the extrinsic reward, and so their intrinsic motivation was hampered.[5]

Once our intrinsic motivation is undermined, it doesn't come back, and we start to look for the extrinsic rewards every time. Even worse, once we're used to the rewards, they stop working. Think back to when you first got a job. The paycheck seemed spectacular compared to the lower income you probably lived on before. Heading to work was, therefore, a big deal because there was that large check every month. But soon enough, the large check just seems like a normal check, and it's not motivating anymore. The only way to get that motivation back is to up the stakes and get an ever larger paycheck. But eventually, just as with the smaller paycheck, you'll get used to that too. It never ends.

[3] http://selfdeterminationtheory.org/browse-publications lists applications of SDT to areas such as education, health care, organizations, psychopathology, psychotherapy, and sport.

[4] Mark R. Lepper, David Greene, and Richard E Nisbett, "Undermining children's intrinsic interest with extrinsic reward: A test of the "overjustification" hypothesis," *Journal of Personality and Social Psychology* 28.1 (1973): 129–137.

[5] It is worth noting that this doesn't occur when there is no intrinsic motivation to perform the task in the first place; paying a child to take out the trash doesn't undermine his intrinsic motivation to do it, as he had no motivation to take the trash out in the first place.

Here again, is another aspect of gamification that is of concern. Because so much of it relies on those extrinsic motivators, you have to keep upping the rewards in order to keep users engaged. This is an arms race you can't win! Eventually, there is a limit to how much you can meaningfully offer, and users will no longer find the offers interesting. Now you've hit the limit of what you can do with your extrinsic motivators, but all the while CET tells us you've eroded the audience's intrinsic motivation as well. With no source of motivation left, they're likely to leave the app soon after. This is another reason why irresistible apps focus on intrinsic motivation: an intrinsically motivating task can carry on for days, months, and years, just for the joy of it. Take, for example, the audience of *World of Warcraft,* many of whom have played the game for a number of years.

A Multifaceted View: Reiss

Thus far, intrinsic motivation has been described as a single value. The theories of Malone and SDT indicate what may move the needle backward and forward on how much intrinsic motivation we have to do a task. Steven Reiss proposed a multifaceted approach that takes into account different peoples' needs at different times. He takes issue with the idea that there are certain tasks that are intrinsically enjoyable to people. Take hiking. He notes that even the most ardent hiker won't want to go out if she is tired, and suggests that the hiking itself is not the goal, but the satiation of the specific need to exercise.

His approach defines a theory of 16 basic desires, which he links to evolutionary psychology. These are listed in Table 2-1, with possible sources of confusion cleared up in Table 2-2. Reiss presents a number of empirical studies to argue for the specific 16 he classified, but there are too many to synthesize here, and interested readers should turn to his paper "Multifaceted Nature of Intrinsic Motivation: The Theory of 16 Basic Desires" from 2004.

Table 2-1. *Reiss's 16 Basic Desires*

Name	Motive	Animal Behavior	Intrinsic Feeling
Power	Desire to influence, be a leader, dominate others (related to mastery)	Dominant animal eats more food	Efficacy
Curiosity	Desire for knowledge	Animal learns to find food more efficiently and avoid predators	Wonder
Independence	Desire to be autonomous	Motivates animal to leave the nest, search for food	Freedom
Status	Desire for social standing (includes attention)	Attention in nest leads to better feedings	Self-importance
Social contact	Desire for peer companionship (includes play)	Safety in numbers	Fun
Vengeance	Desire to get even (includes desire to compete, win)	Animal fights when threatened	Vindication

(*continued*)

Table 2-1. (*continued*)

Name	Motive	Animal Behavior	Intrinsic Feeling
Honor	Desire to obey a traditional moral code	Animal runs back to herd to warn of predators	Loyalty
Idealism	Desire to improve society (includes altruism, justice)	Unclear	Compassion
Physical exercise	Desire to exercise muscles	Strong animals eat more and are less vulnerable	Vitality
Romance	Desire for sex (includes courting)	Reproduction essential for survival of the species	Lust
Family	Desire to raise own children	Protection of young facilitates survival	Love
Order	Desire to organize (including desire for ritual)	Cleanliness promotes good health	Stability
Eating	Desire to eat	Nutrition essential for survival	Satiation of hunger
Acceptance	Desire for approval	Unclear	Self-confidence
Tranquility	Desire to avoid anxiety, fear	Animal runs away from danger	Safe, relaxed
Saving	Desire to collect, value of frugality	Animal hoards food and other materials	Ownership

Table 2-2. *Differences Between Similar-Sounding Reiss Desires*

First Desire	Second Desire	Difference	Example
Power	Status	People who are powerful might not desire social status; people who display high social status might not have much power.	Mark Zuckerberg is a powerful man but wears a hoodie and sandals everywhere. Someone who buys an expensive car to show off might not have any power.
Honor	Idealism	People with high honor may do things that don't improve the world.	A soldier involved in a damaging war would have a high honor to his nation but may not be improving society.
Social contact	Vengeance	Some people play for fun, some people play to win.	Competitive fathers who beat their children at sport play for vengeance motives instead of social contact motives.
Power	Vengeance	Powerful people don't always have to step on the throats of others to get ahead.	A leader of a charity organization is probably someone who enjoys power, but is unlikely to display high vengeance.