

# BURN

A Better Way to  
Build Products

David Cancel  
& Matt Bilotti

# DOWN

Brought to you by



**Drift**

# Burndown

A Better Way to Build Products

by:

**David Cancel**

co-founder & CEO of Drift

**Matt Bilotti**

Drift product manager

@ 2017 Drift.com, Inc.  
All rights reserved.

Designs by: Erik Devaney, Elyse Bogacz, and  
Matt Bilotti

Edited by: Erik Devaney



No parts of this book may be used or reproduced in any manner without written permission from the copyright owner and publisher, except in the context of reviews.

# Foreword

## A case study

Let me tell you a quick story about B12, a company that has incredibly high aspirations. They were working to spin out a new product, but their current approach to product development just wasn't cutting it. It was too slow, and too rigid. They felt trapped in the existing frameworks that were out there and were experimenting with some new approaches, all of which couldn't seem to fill the gap.

Then B12's CEO, Nitesh, found out about the Burndown Framework from a series of Medium posts that we wrote on the topic. They started by following the advice in those posts and were eager to learn more, so we set up a time to walk

through every different aspect of Burndown. They kept asking great questions and were yearning for more detail.

Checking in with them after they got Burndown up and running within their team, they were ecstatic to share that The Burndown Framework completely changed their way of dealing with product and helped them reach (and exceed) their company goals.

In the book below, we explore the philosophy of Burndown, as well as all the tactical things you need to do to make it work for you. We'll cover everything from the design process, to handling tech debt, to team structure, and so much more.

Thanks for picking up this book and we're excited to dive in with you. Keep reading to learn how B12 achieved the product development swiftness they desired.



*David Cancel*



*Matt Bilotti*

# Table of Contents

<b>Chapter 1</b> .....	7
<i>The Current State of Product Development</i>	
<b>Chapter 2</b> .....	14
<i>The Burndown Framework</i>	
<b>Chapter 3</b> .....	19
<i>Getting Started with Burndown</i>	
<b>Chapter 4</b> .....	33
<i>How to Organize Product Teams</i>	
<b>Chapter 5</b> .....	37
<i>The Design Process</i>	
<b>Chapter 6</b> .....	42
<i>Customer Research &amp; Creating Artifacts</i>	
<b>Chapter 7</b> .....	46
<i>Dealing with Bugs</i>	
<b>Chapter 8</b> .....	51
<i>Tech Debt</i>	
<b>Chapter 9</b> .....	53
<i>Managing Your Product Backlog</i>	
<b>Chapter 10</b> .....	56
<i>Prioritization</i>	
<b>Chapter 11</b> .....	59
<i>Transparency &amp; Communication</i>	
<b>Chapter 12</b> .....	66
<i>Getting Your Team Onboard</i>	

# Chapter 1

## The Current State of Product Development

No matter how talented your PMs and engineers are, it seems every software company is still plagued by the following...

- Shipping things too quickly and cutting corners, causing longer-term issues (quick ship releases)
- Not shipping fast enough and frustrating sales, success, marketing, support, etc. (big bang releases)
- Not enough transparency into what the product team is building. Stakeholders feel out of the loop or unclear on exactly what's coming or being worked on

- A never-ending game of prioritization and shifting priorities.

If you're like us, then no matter what you've tried, these challenges still linger over your team. But if you're being honest with yourself, you know your team can achieve more.

That's how we felt back in 2009 at Performable, when we first discovered the principles of Burndown. Then when Performable was acquired in 2011, we got the opportunity to battle-test those principles at HubSpot. Now, at Drift, we're perfecting the principles of Burndown and codifying them.

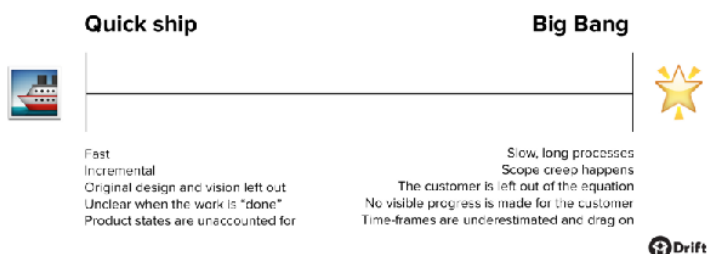
Since implementing the Burndown framework at Drift, we've seen a 5x increase in revenue - far more growth than we ever thought our team could achieve. We hope you will see a similar lift when you finish reading this book.

## **The Quick Ship & Big Bang Spectrum**

Let's talk about the first two major problems listed earlier-- shipping too fast or too slow. In the world of product management, we oscillate between two ends of a spectrum.



Each has its advantages and disadvantages. On one end, we have “Quick ship” releases, which are small, highly incremental just-in-time releases. On the other end, we have “Big Bang” releases. These are giant, long undertakings that attempt to have every detail thought through and perfectly executed.



There are some major problems with both of these approaches:

**Quick ship**—Although this approach is fast and incremental, two things that we love, it often leads to parts of the original product design and vision being left on the cutting room floor. It’s also hard to know when something is technically “done.” Since it’s being built in small,

tiny parts, it introduces new product states that may not have been accounted for in the original design.

**Big Bang**—As the counterbalance to the quick ship releases, big bangs are usually slow, long processes that rarely end up as originally intended. Feature and scope creep takes over and the customer is left out of the process for long periods of time. No visible progress is being made in the eyes of the customer, which leaves the business with a lack of feedback on the decisions they're making. In addition, the time estimates of these releases are almost always underestimated and can drag on for months later than originally planned and budgeted.

With both of these approaches, we end up spending a lot of time playing telephone with each other internally. On the big bang side of the spectrum, things are constantly changing mid-flight, which leaves customers, engineers, designers, execs, marketers, PMs, sales (okay you get the point)...it leaves them all feeling like the product is under-delivering in one way or another. On the other hand, with the quick ship approach, all of the progress and organi-

zation for next steps tends to live in the head of the PM. Milestones are unclear and progress is hard for the rest of the organization to see. A single source of truth is often lost amongst all the quick changes and long cycles because information about different releases winds up scattered amongst all the different tools PMs use: Jira, Trello, Prod Pad, Slack, internal wikis, etc. We all know how messy it can get.

At Drift, we're always looking for ways to craft better processes, and over the past year we've developed a new approach to building software and product.

We call it Responsive Development. It breaks your product development out of the existing spectrum and lets you move more quickly, while increasing your ability to focus on the customer.

We believe Responsive Development matches the way people want to live and work, and that the outcome is a better, more useful, and more reliable product that customers adore.

**What are the first principles of Responsive Development?**

- **It's flexible:** It's based on principles, not rules. Rules are binding. Principles favor progress and forward momentum.
- **It's customer-driven:** Always gather first-hand feedback, not second-hand feedback. Get engineers and designers talking directly to customers.
- **It's iterative:** The first attempt is always wrong. You must iterate towards the best possible solution, not try to get it on the first try.
- **It's rapid:** Speed gives you more iterations and more opportunities to learn.
- **It's visual:** Plans, progress, and updates are done with screenshots and visuals so everyone can see what the end result will look like.
- **It's transparent:** Everyone at the company has access to see what's underway.
- **It's incremental:** The power of progression and step-by-step improvements lead to better results.

- **It's always evolving:** Learning is never done, this is a living framework and will change over time.
- **It favors data over internal opinions:** Results should be measurable and more highly regarded than internal opinions.
- **It's focused:** Iterations should be focused, self-contained, and have as few dependencies as possible.

# Chapter 2

## The Burndown Framework

Burndown is the framework that Drift uses to employ the first principles of Responsive Development.

For example, we use Burndown to make sure we're always gathering customer feedback on an ongoing basis. We feel the customer and end user's pain and make sure everyone on the team (including engineers and designers) are talking to customers on a daily basis to get that first hand feedback. We measure customer adoption, usage, & retention on a daily basis.

We believe that an iterative model that evolves quickly based on feedback wins. The speed in

which we're executing matters and Burndown encourages keeping that bar high and never settling for less. When you look at how easy it has become for competitors in a crowded market to quickly copy one another, you realize how vital speed truly is.

Because consistent reprioritization is key, we try to ensure that at any given moment we're building only what will have the biggest impact on our business's key results over time.

Most importantly, flexibility is more important than stability when it comes to execution. The needs of the customer and the market change rapidly. The framework is flexible to account for constant changes.

Since Scrum is the prevailing method for software development these days, let's see how it compares to Burndown:

	Scrum	Burndown
<b>Measure of success</b>	Thoroughness of story, scrum points and velocity	End user feature adoption & retention
<b>Means of determining prioritization</b>	Product backlogs and sprint planning	End-user validated design mockups and prototypes.
<b>Speed</b>	Story-based sprints (weeks)	Micro-sprints (days)
<b>Release focus</b>	Multiple features grouped into a single version release	A single version of a single feature per release
<b>Flexibility</b>	2-week sprints are planned, executed & generally inflexible once agreed upon	Every day priorities change and so do the current and upcoming sprints

We've been using the Burndown Framework at Drift on a daily basis for over a year now and we've found the results to be spectacular (and our customers feel the same, too). As a result of Burndown, our team is constantly focused on the customer and we've developed a culture where speed matters and flexibility is a foundation of how our business operates.

## Is Burndown Right for My Team?

If you're like us, you put your customers first. Always. At Drift, we're building tools to make their lives easier, and we want to make sure we always stay laser-focused on that goal.



In order to implement Burndown, you must have buy-in from everyone who is on your team. They need to see the value in speed and flexibility over rigid plans.

It's important to keep in mind that things like roadmaps and binding sprint cycles that come along with a framework like Scrum only solve for internal process. Burndown, on the other hand, is specifically built to match what the customer wants. We're willing to sacrifice the month-by-month stability that comes along with something like Scrum for the speed and flexibility that comes with Burndown. (And you should too.)

If you work in a large, corporate environment, adopting Burndown can be tricky (if not impossible). Gonna be brutally honest here...the more process your company already has in place, the harder it's going to be to switch.

**Companies and teams that would be a good fit for Burndown have the following traits:**

- You truly care about your customers and consider yourself a customer-centric business.

- Your product teams are small and focused, no more than 6 members when including engineers, designers, and a PM.
- You're building a software product.
- You, as a leader within your team (no matter what role), fundamentally connect with the first principles of Responsive Development.
- Your company and your product teams value learning and finding new ways to become more productive.

Throughout the rest of this book, I'm going to talk about the gritty details of the tools and processes you need in order to actually implement the Burndown Framework and take advantage of all that Responsive Development has to offer.

# Chapter 3

## Getting Started with Burndown

As the Burndown framework is focused around speed, prioritization, and flexibility, the best tool we've found to manage the process is Trello.

Burndown is a fundamentally visual framework: Images and screenshots are at the forefront of feature development and serve as a way to communicate exactly what will be done in each product release.

Each product team (which ideally consists of one Tech Lead, one dedicated designer, one or two engineers, and one PM) has an individual Trello board. More on that later.

If we were to build a new version of the Drift customer dashboard – lets call it “Dashboard 2.0” – we’d have two options:

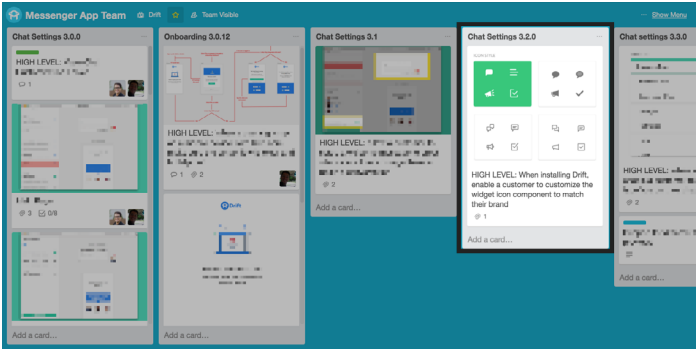
- We could ship the entire thing at once and take two weeks to do that, or...
- We could ship five different incremental updates to customers in two-day bursts.

Which would you choose? To us, it’s pretty obvious. Number 2 is the winner. We call those short bursts “microsprints.”

Each list in the Trello board is a microsprint. Each microsprint is based on semantic versioning by area of the product. Microsprints are an iterative piece of a larger feature that can be shipped independently. They are contained pieces of work that could take anywhere from 30 minutes to 2–4 days. If anything is going to take longer than that, it needs to be revisited to see if it’s worth tying resources up in something for such a long amount of time (yes, 4+ days for a single new feature or product update is a long time in the world of Burndown).

The Trello board is organized from left to right. Next-up releases are the furthest left on the Trello board. As you scroll to the right, you begin to see future releases that are still being explored and designed.

In this example below, the upcoming micro-prints were focused around onboarding and settings.



NOTE: For a while when we were developing this framework, we relied on versioning of each release (ie: release x.3.x), as shown above. Over time, we realized that the upkeep of this was causing the team to stress too much about maintaining number consistency, so we moved away from that as part of the Burndown process. If you're concerned heavily with the ability to historically have a record of what was done when, you may want to consider maintaining versioning.

At the top of each list, there is a HIGH LEVEL card that explains exactly what is going to happen with that release, written in the format of a Jobs To Be Done statement. The HIGH LEVEL card also contains a link to the Sketch file that has all of the designs for that release.

For those unfamiliar with Jobs To Be Done (JTBD), the goal of JTBD is to focus on the intended outcome, rather than the actual product or service itself. An example is that a customer may buy a power drill, but the real thing they want is a hole in their wall to hang a painting.

The Job To Be Done phrasing of this situation, from the perspective of the customer, would be “When I want to decorate my apartment, I need a way to drill a hole in the wall so that I can hang a painting.

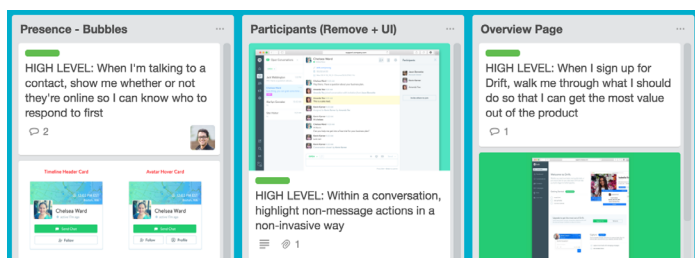
Using this system lets anybody in the company, at any time, see exactly what the outcome of each microprint will be. Transparency is key to the Burndown framework because priorities are constantly shifting and everyone (execs, sales, success, support) needs to have the

ability to keep up with what's coming down the pipeline next.

Each HIGH LEVEL card has an engineer who owns the card. That engineer is responsible for playing quarterback on the release and working with the rest of their team (and other teams) to make sure all dependencies are accounted for and the end result is as expected. Engineers assign themselves to HIGH LEVEL cards when they're actively working on features.

As you start scrolling to the right on the Trello board, you'll come across red labels, which are used to signify that design work is still needed for that list. Designers work from left to right, focusing on the left-most list with the red labels. Once design is done, there's a conversation between the designer and whichever engineer is likely to work on the feature to make sure everyone is on the same page with the intended end-result. PMs are usually not present during these meetings. The designer then crafts the HIGH LEVEL card, based on the JTBD that the PM wrote in an "artifact" (an in-progress collection of research...explained in a later chapter) and breaks that out into individual cards within the list. They remove the red

labels, and move on to the next list to the right with a red label.



Managing these lists in Trello allows us to be super flexible with customer feedback. As first-hand feedback comes in, the team is empowered and encouraged to re-consider the prioritization of the upcoming lists. It's an on-going conversation. Even though I, as the PM, might put X before Y on Tuesday, it's totally fair game for a designer or engineer to make a case on Wednesday as to why the order needs to be swapped. Then, it's as easy as dragging and dropping a list on your Trello board. Responsibility is shared.

## Labels we use

There are only a few labels that we have on our Trello cards. Here they are...



Dependency on another team

In order to get this feature or update shipped, work on another product team must be completed. It's up to the PM and engineer that owns this card to coordinate.

Design needed

This card is defined using the job to be done framework and now requires a designer to make it real.

File Breakdown Needed

After something is designed and has gone through a few rounds of testing, the designer throws this label on a card that needs to be broken out into a full list.

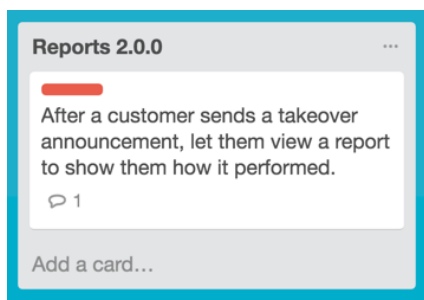
## **Design and engineering are responsible for the details**

Product management's role is to gather vast amounts of customer feedback and great examples of how others solve the challenges that customers face today. This means that the product manager is defining the high-level goal (the JTBD) of each list, rather than trying to figure out all of the design and engineering-focused details. They're responsible for filling the list with screenshots of customer feedback, market research, artifacts (explained later), and more context for the rest of the team to utilize. Those are up to those teams to figure out together, which increases ownership on the product teams.

As an example of this, I, as a product manager, would create a column for the high-level need

of the list. Let's use a specific report as an example.

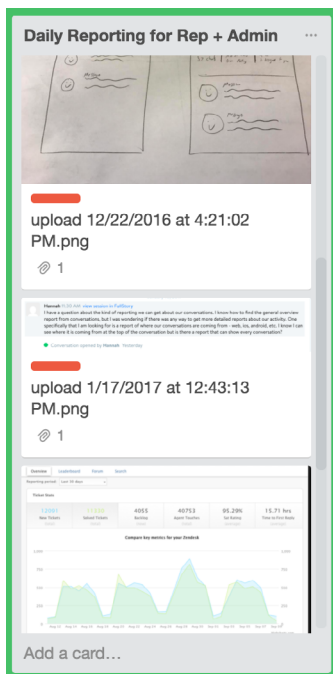
Let's call this column "**Reports 2.0.0**".



In the description of that card, I'll put something along the lines of:

"Now that we are rolling out a new messaging style, we need to make sure our reports can support the data and results from that message. These reports need to handle displaying percentages, raw numbers, a chart, and profiles."

In addition to that, in the comments on that card I will post notes from conversations with customers, screenshots of feedback they sent via in-app chat, and some examples of other products that do an amazing job with this sort



of feature. The idea here is volume, not curation.

Once the designer reaches this column as the next thing they're tackling, they sit down with product management to talk through the high-level needs on the card and any potential solutions based on the feedback and data that the PM has col-

lected and added to that card over time (AKA: "Artifacts" - discussed in a later chapter). From there, the designer will check in with engineering to get a sense of the limitations and constraints and circle back with the PM if necessary.

After talking things through with engineering and product management, the designers take a first pass at a solution, which they then turn into an Invision file with a clear walkthrough of the update. That Invision file is shared inter-

nally, as well as with a handful of random customers as well as ones that have requested it in the past.

If the reviews are overwhelmingly positive, design will sweat through any details with engineering and break out the HIGH LEVEL card in the Trello list. Each of these Trello lists is what we'd consider a microsprint - something shippable on their own that adds value to the customer. If it doesn't add value to the customer, it can't be considered a standalone microsprint. If, on the other hand, there was a lot of constructive feedback, they will take another pass and repeat the process. For a major release, such as rebuilding reports entirely, this cycle may take 2-5 days. The important thing to note here is that this does not mean that engineering will implement the whole design at once. That's the point of microsprints: They're iterative.

## **Breaking out the HIGH LEVEL card**

Once design and engineering are happy with the designs and we have a thumbs up from the customers, the designer will then work with engineering to break out the HIGH LEVEL card

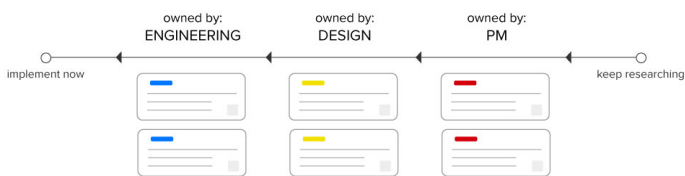
into individual cards. The designer is responsible for making sure each microsprint is a cohesive step towards the full end result that can have a solid UX on its own, and the engineer is responsible for making sure each microsprint is attainable and can be shipped on its own within a max of a few days. As mentioned earlier, the HIGH LEVEL card will contain the overall sketch file, but then each piece of the UI or functionality that is in that release will get its own card with an attached screenshot. This is important because it makes sure that small things don't fall through the cracks and all the details are thoroughly considered.

If the feature release is a larger one, the HIGH LEVEL card will be broken into multiple different lists so that each list is still a microsprint. This is one of the things where Burndown is highly different than something like Scrum. Microsprints are small, incremental releases that work as stand-alone product updates. This leaves room for other higher-level priority work to happen in the midst of reaching the fully expected UI, while also making sure that things aren't pushed aside and forgotten about.

As engineers complete different pieces of a mi-

crossprint, and the individual things are shipped to production before the entire release, the individual cards are archived. Once an entire list has been shipped to production, the engineer leading the HIGH LEVEL card then archives the entire list and moves on to the next thing.

### **How lists are changed, updated, and prioritized (and who owns what)**



Engineering is usually working on the first 2–3 lists depending on the team size. Individual engineers should be assigned to no more than one HIGH LEVEL card (AKA one list) at a time, as assignment is used to signify who is working on things at that moment.

At any given time, design is about 5–15 lists ahead of engineering. If they're consistently further ahead than that, it means you should reconsider the size of your releases or how you're prioritizing. Just because they're ahead doesn't mean engineering should be imple-

menting everything they've done so far, in that order. That goes against the fundamental principles of Burndown.

Product management is responsible for owning the priorities of lists 5+. They are working closely with design on 5–15 to make sure the right things are up next based on customer (and business) needs. Design mainly owns those cards while fleshing out the final design and breaking out each release into smaller pieces. Throughout this process, they are validating the designs with customers (unless the update is extremely minor).

Product management is spending the majority of their time on lists 15+ to validate the hypothesis of what should be worked on in the coming days/weeks. To the far right of the Trello board is where cards are consistently added, archived, and updated. It's the breeding ground for what's coming down the pipe and needs more customer validation.

The most important part of all of this process is that it A) removes the need for a project manager and B) removes the need for the product manager to continually play project manager

throughout this process. The product manager is a knowledge worker and should be doing things like talking to customers more than they should be doing things like figuring out every detail of how things will be solved and implemented.

### **Nothing is set in stone, ever**

There are cases when we get customer feedback or make a strategic shift that re-organizes our priorities, including cards 1–5. While it may seem chaotic, and it sometimes is, the order of the lists change every single day to keep us working on what matters most. This is the most fundamentally important aspect of Burndown.

The result is a constant focus on the most high-leverage items possible, which enables the team to ship maximum value to customers that align with business goals.



# Chapter 4

## How to Organize Product Teams

While it's not required to organize your teams a certain way in order to use Burndown, we wanted to share our philosophy on ideal team structure.

There are a few key principles behind the foundation of our product teams:

- **Each team owns a customer-facing portion of the product.** Each team must have something they can point to and say “that is what the customer sees as a result of our collective work.” These teams need to be able to ship customer-facing products without any dependen-

cies on any other teams.

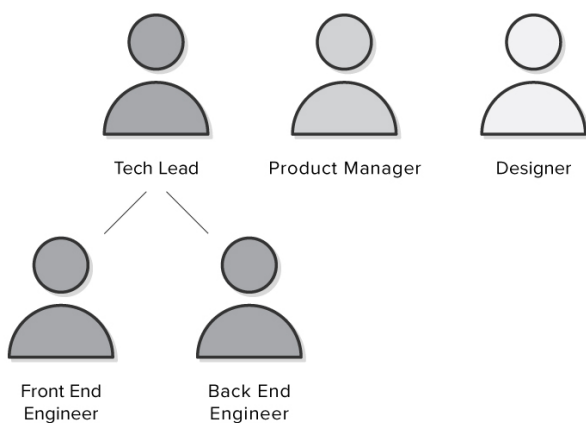
- **While splitting by code base is nice, it's not always ideal.** We prefer to have teams cross-collaborating on a code base if it means that each team can ship independent value. Splitting teams by code base can do wonders to encourage code ownership. This breaks if, for example, you have only one team working on backend - that team then becomes a resource fought over by other teams and can become a bottleneck for shipping product updates.

- **Some roles will be on multiple teams as you scale** (and can help you scale) Your first PM might wind up on three teams and stretched thin until you make your next PM hire. That's okay. If you spin out a new product team, a designer might start working across two different teams. That's also okay. It helps the organization scale and gives you clarity into where to make the next hire based on current or foreseeable bottlenecks.

**The ideal team makeup consists of the following dedicated resources: one product manager, one designer, one tech lead, one front end engineer, and one backend engineer.** This

makeup allows teams to be nimble and each team member to have clear ownership on their part of the team's product. Note: At scale, an ideal team also includes a dedicated product marketing manager.

### **The Ideal Product Team**



#### **Product Manager Responsibilities:**

- Understanding the market
- Framing the problems the team is solving
- Guiding the overall prioritization

#### **Designer Responsibilities:**

- Crafting the solution alongside the PM & Tech Lead

- Iterating on the solution with customers
- Making sure the UX of what is delivered matches what is intended

### **Tech Lead Responsibilities**

- That the solution delivers upon solving the problem
- That the engineering team is shipping
- That the product works
- Building new features, fixing bugs

### **Front End Engineer Responsibilities**

- Building new features, fixing bugs
- The things they ship work

### **Back End Engineer Responsibilities**

- Building new features, fixing bugs
- The things they ship work

### **Product Marketing Manager Responsibilities**

- Being the bridge to marketing and sales
- Crafting content and collateral to launch and market the solutions the team builds

# Chapter 5

## The Design Process

Nailing the design process in Burndown is an evolving goal. We've gone through a few iterations in the past to get to where we are today. It's taken a lot of grinding to learn the best flow here. It's also important to note that this process may not fit perfectly with your own team's structure and can depend on both the type of designer you hire and the mindset of your Tech Leads.

There are three stages of the design process within Burndown ([took some lessons from Dropbox's process here](#)):

**Stage 0** - Defining the Job and defining success. This is where we (the product managers) frame the Job To Be Done (JTBD) that we're trying to solve for and the metrics we associate with it being successful once launched. It's also when the product manager, designer, and engineer sync up to look at the artifacts the product manager has been creating to understand what the technical limitations might be.

- A good example of a Job To Be Done is: "When a lead comes to my website, I want them to talk to the most relevant sales rep (or a CSM/Support person if they're already a customer), so that I can give each lead the attention they need, in a timely manner, and close them as customers."
- A bad example of a Job To Be Done is: "Add lead routing into the product to allow people to route new chats by geography or other attributes."
- The good example is product agnostic. It doesn't matter if Drift is creating the solution or not. This is simply what the customer is trying to achieve...the job they're trying to get done. The bad example defines a solution

before acknowledging the actual intent of the customer.

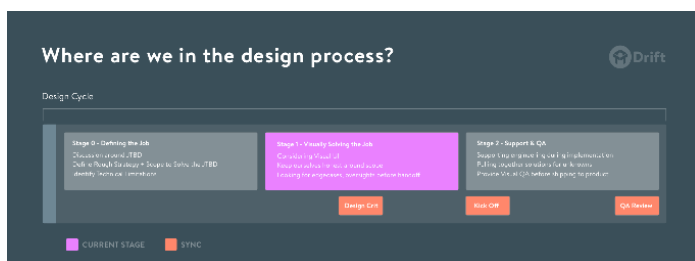
**Stage 1** - Visually solving the job. This is when design takes what they understand of the problem, uses the customer research, talks to customers themselves, and starts to create a visual solution for the job we're solving. This process may go through a few cycles of internal and customer iteration. The bigger the update to the product, the more iterations it'll go through.

- During these cycles, there will be at least one Design Crit, where a fellow designer from another team looks at the solution the designer is crafting and gives feedback.

**Stage 2** - support & QA. This is when the solution is actually being implemented. The designer and the engineers building the solution spend time together to make sure what's being implemented: adds value to the customer, solves the JTBD agreed upon, and works as intended from a UX/UI perspective.

- Kick-off: This stage always starts with a Kick-off. This is a 5-10 minute sync up between ev-

everyone who will be implementing the solution (the engineers, the PM, and the designer) to make sure everyone knows exactly what they'll be doing. We once had a problem where people would start implementation without fully understanding the exact details of what they were trying to accomplish. The Kick-off surfaces any friction or unanswered questions. We go around the room and each person says "I'll be doing X, Y, and Z for this." The goal of this meeting isn't to assign work or to discuss the details - that should be done in Stage 1. The goal of this meeting is to verbally make sure everyone is on the same page about what happens next.



It's also important to note that the designers always start by designing the most optimal solution. Once they have that and the team has agreed on it, they will sit down with the engi-



neer who will be implementing the solution to figure out how to break it into microsprints.

*I'd like to give a hat tip to Elyse Bogacz, who is our rockstar design lead that has helped nail down this process over hardcore iterations the last 6-12 months.*

# Chapter 6

## Customer Research & Creating Artifacts

As noted in Stage 0 of the design process, it's important for the team to agree upon the job that is being solved. Since it is the product manager's job to frame the solution and to collect customer feedback, product managers need to be creating artifacts.

**An artifact** is simply documentation on a the problem, market research, a collecting of pointed feedback from customers, or possible suggested solutions. Often, these take the form of a collection of screenshots, lists, Job To Be Done statements, wireframes (pen/paper, whiteboard, software...doesn't really mat-

ter), flow charts, etc. It is what helps define the scope of a potential V1 for solving the job to be done.

### Artifact Example

**Job to be done:**

Here's where customers are struggling:

*(screenshots of customers talking about their pain)*

Here are some possible solutions:

Here is a suggested UX:



In addition to dumping any/all feedback related to a specific topic in a Trello card to the far right of the Trello board, the product manager should be creating larger artifacts. At Drift, all of those are posted in an internal Wiki (we use Confluence) and are shared in Slack with the

team well before you actually enter Stage 0 of the design process. You could also create these in Google docs or other documentation tool you use. The important thing is that it's somewhere that's easily referenceable and accessible in the future when you do get to stage 0.

Artifacts often look different for different undertakings. There is no "right way" to build an artifact, so long as it's useful to the team.

Ideally, before anything enters Stage 0 of the design process, the product manager will have created and shared 2-3 different artifacts that the team has seen and commented on.

The goal of the artifact is to create more context for the design and engineering teams when they sit down to craft the perfect solution. It may have suggested solutions, but it is never what is usually known as a spec or requirement document.

An artifact takes the perspective of "here's what we know, here's the data, here's the frame of the problem, here are a few possible solutions, here's what competitors are doing..." rather than your typical perspective of "The solution

must include X, Y, and Z. Here is a wireframe of said solution, etc.” that a normal spec or requirements doc has. We don’t like those -- it is not the product manager’s job to create the solution. It is their job to frame the problem and provide the highest amount of firsthand feedback and context as possible so the rest of the product team can own and implement the solution.

# Chapter 7

## Dealing with Bugs

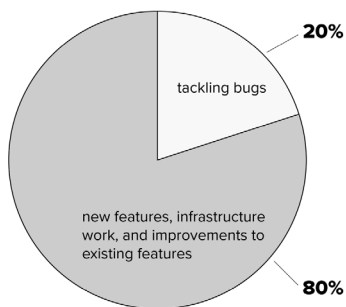
Since we're dealing in contained, continuously shipped microsprints, it's much easier to avoid major bugs in the code because you're being gut-checked in production on basically a daily basis. If bugs do arise, they should be relatively easy to tackle because you're likely to know what caused them, unless they go unnoticed over the course of many microsprints and surface way later.

We believe bugs should be entirely owned by the engineering team. As such, Github issues are where they live. When bugs pop up, anybody on the team can create a Github issue. If

it's a customer-reported bug, we give it a red "Bug" label in Github and post a link to the conversation where it was reported. We prioritize these bugs over others.

As a general rule of thumb, we're spending about 20% of our time tackling bugs, and the other 80% on new features, infrastructure work, and improvements to existing features.

### Product Team Focus



As bugs surface, it's up to the engineer's discretion which need immediate attention and which we can hold off from for a bit. This is why the "Bug" label in Github is so important: because it increases visibility and urgency into the fact that it's happening for a paying customer, which could result in churn if the bug is critical.

While we were implementing Burndown, for the first few months, once a week, the product manager on each team would sit down with the tech lead and go through the backlog of bugs in the Github repos that their team owns. This way, the PM and the engineers could get aligned on what was most important, why, and learn to trust each other's judgement over time.

As the Burndown principles and framework became more a piece of the culture, and as the PM would trust the engineers to make the same calls they would, that meeting was no longer necessary on a weekly basis. Now, at the start of each week (usually over the weekend) the tech lead and engineers will go through the Github issues and pick out the top most important bugs they plan to tackle that week.

Since Burndown doesn't operate in hard week-long cycles, there are consistently bugs that are introduced that require us to halt what we're doing and tackle them right away. So we make sure to leave extra capacity to handle those as they come up so they can be worked on in parallel.



Recently we received a few questions about what our QA process looks like. As mentioned in the earlier chapter about the design process, QA is done by the engineer, designer, and PM before the product is shipped to production. Often it's as simple as the engineer saying "Hey Amanda [the designer], can you come over here and take a look at this with me to make sure it looks and works as expected?" In addition to that, we use Github and engineers share their pull requests for somebody to take a look at before it's pushed.

We've looked into many of the automated QA systems out there, and most of them are excessively expensive and cause your entire shipping process to slow down. Since one of Burndown's edges is speed, this is catastrophic to the flow. In addition, we've seen that using these systems often alleviates the engineers from truly owning their code. It's way easier for them to feel like "ah, whatever, somebody else will catch it if it's broken" rather than being fully responsible for making sure it works in the first place. No, we don't write unit tests, but we expect engineers to own the code they're building to make sure it works.

In addition, we've used other automated front-end testing frameworks, and we've found that often they're far too brittle and the upkeep of those actual tests is more than effort than expected, especially as the product surface area extends rapidly.

We do a few additional things around QA. Here's our suite of tools to avoid bugs in the first place:

- We've baked QA into the design process
- We use Assertible to test our endpoints
- We've begun using Segment's Nightmare framework for critical tests (The onboarding works, login works, etc). The great thing about this is that we can actually hook these tests into our build process (using Travis) and fail the build when the test fails
- We have a QA tester that we hired via Upwork. He is on-call always and does daily live tests of critical functions, as well as monitors our #shipyard room in Slack where we post all of the things the engineer ship, as they ship them to test anything new.

# Chapter 8

## Tech Debt

Alongside the conversation about tackling bugs inevitably comes the question of “well, what about tech debt? When and how do you tackle that?”

There are a few things to keep in mind here. The most important is that **the customer does not care about your tech debt. At all. It is irrelevant for them.** Tackling tech debt is often a deep, dark rabbit hole. Something that the engineers pitch as a 2-day thing inevitably turns into a 3-week re-write of the entire infrastructure. We’ve all been there.

Here's our rule about tech debt - It is only tackled when tied to customer value. Tech debt will not be touched until it is tied directly to a new feature that is being rebuilt. An example of this might be that the onboarding flows were hacked together 6 months ago and the code base is super janky. Let's say there are small tweaks you want to make here and there. Of course, the engineer is likely to want to rebuild the whole thing because they're playing with a jenga set, but changing one step isn't actually adding much customer value. In this case, we wouldn't tackle any onboarding-related tech debt unless we were to completely revamp the onboarding flow to be more useful.

# Chapter 9

## Managing your product backlog

By now you might be wondering: What do you do about your backlog that has ideas for 4, 6, and 12 months out? Where do you keep all of that information and keep it organized?

Depending on the size of your backlog, the Trello board you use to implement the Burn-down framework may begin to break from too many lists on the far right of the board, which is completely fine. That's what happened to us. So we evolved.

Now, we use the main product Trello board only as a place where we scope and document

microsprints that will actually get done. If something's been in there for over a month and keeps getting pushed around and never done, that means it's not top priority and likely will never be. So we archive that list.

The Burndown framework is designed to keep you focused on what's in the short and near term around the highest-leverage stuff you can be doing. Anything designed and scoped that's 3+ months out, unless you're a larger, extremely structured organization, is often going to change drastically by the time you reach that time anyway. We've had a designer on one of our teams get 3+ months ahead of the team and ultimately most of what she designed never made it to the light of day because we learned quickly that we wanted to take things in another direction. And that's okay. That's what we sign up for using Burndown. The worst-case scenario there would have been to continue implementing those designs simply because they were already done. Since Burndown is inherently flexible, it allowed us to easily say no and take a different course that had more promise.

If you're like me and are customer-develop-

ment focused, you still need a place to organize all of the customer feedback you get on a weekly and daily basis. I'd suggest either making a different Trello board to hold all of these ideas and pieces of feedback or keep expanding the width of your Trello board indefinitely so long as you go through and gut-check it every few weeks and archive things that are not aligned with where the product is going.

# Chapter 10

## Prioritization

The best way to determine what the next most important list to tackle is to ask yourself this...

*“What is the one thing that provides the most value to our customers and aligns with our business goals and focuses?”*

Answering this question should provide you with the thing that will give you the most leverage in the long-term to continue adding value to your customers. The question is a derivative of the question that Gary Keller poses in his book, *The One Thing*, which is “What’s the one thing I can do right now to make everything



else easier or unnecessary?”

We're big fans of this philosophy at Drift as we believe it helps us focus on the right things.

We also thematically structure the upcoming priorities of the company, which we put into a single slide or list and repeat over and over again to the team. In our case, an example list would look something like this...

- Keeping the product stable
- Driving more signups
- Increasing the activation rate
- Innovating our chat widget

This gives us all a guideline to reference when we're asking ourselves what we could be focusing on next. If you can't do something that directly benefits #1, then work on something that benefits #2, etc.

The power of microsprints really shines when it comes to prioritization. Instead of locking ourselves into, let's say, two full weeks of work on a new dashboard, we can commit to five two-day microsprints. After we get through the first two microsprints (that add customer-facing value on their own), we may shift priorities as

a business around one core goal. Let's say we learn that what we've been building isn't useful or we simply come to recognize that working on something like billing gives us way more leverage. Because we broke things into micro-sprints, we can easily table the next three micro-sprints for the dashboard and instead focus on that new high-leverage work, thus coming back to the dashboard later.

# Chapter 11

## Transparency & Communication

Since the Burndown framework is one that constantly shifts as the microsprints inside of it are continually moving to match the needs, wants, and desires of the customer, it's important to make sure the team is on track and has milestones to hit.

Burndown doesn't subscribe to the concept of a roadmap with deadlines. We fundamentally believe that roadmaps only serve company needs, not customer needs.

So here's how we stay aligned on priorities...

Every Friday, the product manager will sit down with the tech lead of each team (or the entire team, depending on that product team's culture) and talk through the upcoming micro-sprints for about 10–15 minutes. Sometimes, it's clear what the next most important thing is and we're immediately on the same page. Other times, good points will be brought up in the conversation and microsprints/lists will be shifted a bit and agreed upon as the focuses for the upcoming week. During that meeting, the product manager writes down exactly what each engineer and the designer is committing to for that upcoming week. Then they'll all review that list before leaving the room to get in agreement.

Each Friday afternoon, the product manager on each team then writes an internal wiki post with a tactical breakdown of which lists from Trello will be focused on during that upcoming week and what value they provide to the customer. That gets posted on Sunday evening in Slack. This internal post generally takes about an hour total to write, but it's 100% worth it to keep the team aligned as they show up the next morning. If done right, this is the only project management that a product manager would


need to do on a weekly basis.

**Here's the structure of those posts (one per product team each week)...**

1. A brief recap on the week prior, a list of what the team shipped the previous week, and a high-level list of the themes that are being focused on this next week
2. A breakdown of what each individual on the team will be doing that upcoming week
3. Visuals/screenshots (from the HIGH LEVEL cards in Trello) of what will be shipped
4. A visual representation of the upcoming themes

At the top of this internal post (#1 in the list above), there will be a few sentences on what themes were completed last week, and a rough outline of what's coming next week. Here's an example:

## 2.14.2017 - Everywhere Team Weekly Product Update

Created by Matt Biloti, last modified by Amanda Yee on Feb 12, 2017 •  Similar Content

Alrighty. Making huge progress on billing/trials/pricing. Almost there.

Admin roles are underway. Let's get them shipped.








Big plans for improving all of our signup, auth flow stuff. Let's get it.

Here is what we shipped last week:  [Weekly Shipping Report](#)  [Everywhere Team]

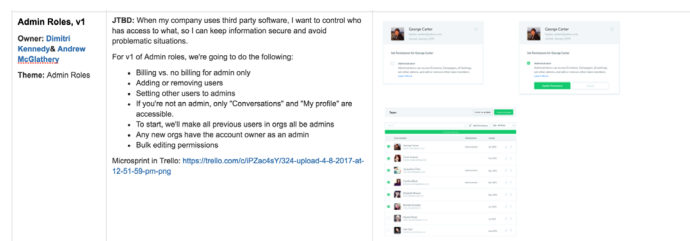
### Main focuses for this week

- **Billing (Growth Team)**
  - In-product pricing to match /pricing. Goal is both are launched by EOW
  - Ability to change credit card
  - Bugs in billing
  - Grandfathered plan
- **Trial Updates (Growth Team)**
  - Self-serve trial
- **Admin roles v1:**
  - Backend support and infrastructure
  - 1 new role
  - Billing vs. no billing for admin only
  - Adding or removing users
  - Setting other users to admins
  - If you're not an admin, only "Conversations" and "My profile" are accessible.
- **KTSR:**
  - Revisiting all auth flows
  - Additional polish
  - Live View Scalability
  - Any bugs that come up

Then, for part #2, the product manager takes the things that were agreed upon in the Friday meeting and puts them into a table. You'll notice that each person on the team has a main focus -- that's their one thing that's more important than anything else on their list for that week.

EVERYWHERE TEAM			
		 1 Thing  Bug	
PETE	DIMITRI	ANDREW	AMANDA
 Upgraded pricing in product	 Admin roles, v1, shipped	 Admin roles support	 Routing (Lead assignment) v1
<ul style="list-style-type: none"><li>◆ Start Trial from billing page</li><li>◆ Any billing bugs</li><li> Handling grandfathered plan</li><li>◆ Changing a credit card UI (stretch)</li></ul>	<ul style="list-style-type: none"><li>◆ Auth flows</li><li>◆ Polish for things shipped in the past 2 months (notifications, invoices, onboarding)</li></ul>	<ul style="list-style-type: none"><li>◆ HelpScout Integration needs</li><li>◆ HelpDocs Integration update</li><li>◆ Live View Scalability</li><li>◆ Billing launch support</li></ul>	<ul style="list-style-type: none"><li>◆ Admin roles v2</li></ul>

For part #3, each of the main focuses is broken down alongside the Job To Be done and a link to the microsprint in Trello. Since Burndown is visual, screenshots of what will be released in that microsprint are added for everyone to see what (in an ideal world) will be in the product sometime over the course of the next week.



Part #4 is a high-level overview of the entire product organization's upcoming priorities. This is for the execs, sales, success, and marketing teams to see what's roughly coming down the pipeline at a higher-level. The below example theme document is from a custom-designed Sketch file. Once you scale past a few product managers, you'll need to look into managing this off of a collaborative tool such as Google Sheets or Roadmunk. That way it can be linked to at all times and you don't have to worry about multiple PMs trying to edit/up-

date this document at the same time.

## Product Priorities - 12/23

	EVERYWHERE TEAM	CHAT TEAM	APP TEAM
MAIN FOCUS	<p><b>NOTIFICATIONS</b></p> <p>📌 Intelligent Notification Fallback</p> <p>✔️ Updated Settings UI</p> <p>✔️ Notification email design simplification</p> <p>Slack command for debugging notifications</p> <p>Cascading/continual sounds</p> <p>Grouped email update</p> <p>Drift Web/Mobile App Presence (to detect Activity)</p> <p>🕒 Mirror Updated notification settings in iOS + Android</p> <p>Better browser notification logic (don't send for convos you're on)</p> <p><b>SALESFORCE INTEGRATION</b></p> <p>✔️ Passing new Drift contacts as new in SFDC</p> <p>Passing conversation transcripts to SFDC</p> <p>Submitting a basic package to SF for security review</p> <p><b>PRESENCE</b></p> <p>UI to view presence and start a new conversation</p>	<p><b>PRESENCE SUPPORT</b></p> <p>📌</p> <p><b>SUPPORT VOLUME CONTROL</b></p> <p>Route free users to a different inbox</p> <p>Route paying customers to paid inbox</p> <p>✔️ Route free users to offline flow only</p> <p>Route site visitors to sales inbox</p> <p>Make free user submissions go right to email</p> <p><b>K.T.S.R. - CONVERSATION UPDATES</b></p> <p>Plain text fallback email</p> <p>Emoji menu (because it needs to happen and we've been putting it off too long)</p> <p>📌 Campaign as first message in conversation</p> <p>Welcome message</p> <p><b>K.T.S.R. - SETTINGS &amp; PROFILES</b></p> <p>Re-org settings into tabs</p> <p>Adding Facebook &amp; portfolio links</p> <p>Bugs with redirects and pages</p> <p>Load photos from CDN (they're super slow)</p>	<p><b>BOT FOR EVERY WEBSITE V1</b></p> <p>Bot Dialog UI</p> <p>Report UI</p> <p>✔️ Edgescence for Bot Dialog</p> <p>✔️ Bot Interaction Editor UI</p> <p>✔️ Support UI Tagging</p> <p>Follow Up Editing - Flag Default, Add Examples, Delete</p> <p>Question Editing - Archive, Confirmation Modal, Reorder</p> <p><b>BOT FOR EVERY WEBSITE V2</b></p> <p>Build a Campaign - Reference a Bot Interaction</p> <p>Bot Interaction Editor - Name</p> <p>Bot Interaction Editor - Blank State</p> <p>Bot Interaction Editor - Preview</p> <p>Bot Interaction Editor - Rich Text</p> <p>Bot Interaction Editor - Clone</p> <p>Bug: Campaign Routing by Inbox</p> <p><b>K.T.S.R. - CRM VIEW</b></p> <p>📌 Filter By Campaigns</p> <p>Filter by ORs</p>
	NEXT	<p><b>FULL BILLING WIRING</b></p> <p>📌</p>	<p><b>MODERN WIDGET, V2. NO OFFLINE SCREENS</b></p> <p>📌</p> <p><b>SCHEDULING CARD</b></p> <p>📌</p>

Remember, within each week there can be many microsprints planned and we are shipping things immediately when they are completed. These weekly check-ins and updates serve more as a way to ensure that everyone has a sense of what progress looks like and a tangible means to know exactly what to focus on when they get in on Monday.

The things posted in the Sunday update aren't a



“this must be done no matter what” kind of list. Sometimes things will happen and things we planned to do are no longer worked on since something higher priority comes up that could better impact our current company priorities/themes. That’s the beauty of the flexibility of short microsprints.

# Chapter 12

## Getting Your Team Onboard

If you're a larger organization with a strongly rooted process, I'd recommend working within one product team to start (a couple engineers, a PM, and a designer).

Get one team working well with microsprints and semantic versioning in Trello, prove it's an efficient way to build products, and then scale it across your organization.

If you're just starting up or building, you should start with microsprints in Trello and bake it into your product culture. Semantic versioning is like a perfect set of training wheels as you

and your team get started with microsprints, so take them seriously and stick with them. When you feel like they're truly getting in the way of your speed, that's when you know they are no longer necessary. For reference, it took us about 3-4 months to get there.

Here's how our semantic versioning system looks...

- **N.x.x (Major Release)** - A major release is when a brand new part of the product is introduced or an existing part of the product gets a complete overhaul.
- **x.N.x (Minor Release)** - A minor release is when a new feature is introduced within an existing part of the product.
- **x.x.N (Patch)** - A patch is when an existing part of a feature is tweaked slightly.

In addition, here's a helpful set of common objections you might get from the rest of your product team if you're trying to move to Burn-down alongside suggested responses...

**The objection:** We already have a process that

works. Why would we change it?

**Potential response:** If you told me that you had \$10 to spend and you were going to store A and instead I told you that right next door we could go to store B where your \$10 can buy you twice as much, would you follow me to store B? Probably. Ultimately, the incremental and weekly benefit we get from switching to Burn-down will pay for itself in only a few weeks and we'll be delivering more value to our customers more often.

**The objection:** How do you know this is going to work?

**Your response:** It's been tested and refined by David Cancel, five-time founder, and the product teams at Drift, HubSpot, and many others. They've surfaced challenges and have a clear framework for how to implement it on our team. There's also an entire handbook on the process that you can check out first.

**The objection:** How long will it take us to get this new process up and running?

**Your response:** About the length of one full

two-week sprint. We'll move all of our backlog into Trello boards over the next few days and we can even start using it today for the things that the product teams are currently tackling.

**The objection:** There are details I think are missing from this framework and questions I have. How would we reconcile those?

**Your response:** Like we always do...as a team. Burndown is inherently flexible and ever-evolving. We can absolutely adapt it to our team and how we work best. This isn't a means of saying "we must operate exactly like this" but more of a means of saying "this way of building product seems better. Let's take the pieces that work for us and improve our own process with it."

**The objection:** I don't use tools like Trello, Sketch, Invision, and Github, etc. Are they required?

**Your response:** You need some sort of tool to manage microsprints. You could do a real-life wall with sticky-notes, but it's not recommended. Almost all of these tools are super simple to get started with. Trello is also 100% free and barely has any learning curve at all.

**The objection:** I work in a larger, traditional, organization where I need to set go to market milestones throughout the year with marketing, sales, etc. How can I do that while also using Burndown?

**Your response:** We almost always publicly launch or release products that we've already rolled out to customers. This allows us to get case studies and quotes from customers already using the tool, and provides marketing and sales with a product that they know will work and that they have validated use cases for.

# Conclusion

In the world of B2B software, consumer software, mobile apps, SaaS, etc. the reality is the days when innovative technology was a true differentiator are coming to a close. Today, most software companies build products using the same tools as everyone else.

One of the best ways you can set yourself apart is through the process you use to build that software. Responsive Development and the Burndown framework provide a more customer-focused, flexible model of building software that can give you and your company an edge against competitors.

At [Drift](#), we're obsessed with helping every company in the world know, grow, and amaze their customers. It only feels right that we make sure the process we use to build our tools follows that philosophy.

Thank you for reading this book on Responsive Development please feel free to reach out to [matt@drift.com](mailto:matt@drift.com) if you have any questions or feedback on this book or Burndown. We're here as a resource for you as you implement Burndown and would love to hear how it goes.

If you enjoyed this book and think some other people may find it useful, we'd be so grateful if you tweet out a link to it.

Good luck, from your friends at Drift.

- [Matt Bilotti](#) & [David Cancel](#)



*Matt Bilotti*



*David Cancel*



