Vibin Labs

A Study On Software Abundance

Part I: Narrower Than AI Boosters Claim and More Real Than Skeptics Realize

Robb Gaynor

April 2026

Executive Summary

People are here to stay! People are using AI to augment their output. In the one domain where AI has shown the greatest strength, software engineering and writing code, a person does 61% of the work to get a result, to build an app. Using AI today is about augmentation. And if this is true in the one area AI excels, it is a telling fact when looking at other areas where AI might be deployed to augment people. This is a story about human augmentation and software abundance.

One person, a product manager, created 33 apps in 75 days. An App Orchestrator, a builder of apps. The products built represented 405,000 lines of code. This is software abundance. A single person, coupled with an LLM coder (Claude Code) can now produce products many, many times faster than the traditional software development process. The software factory process used in this test is not unique and, the process is repeatable.

The world is filled with AI hype. Senators are speculating, CEOs are bailing on their companies, Nobel laureates are predicting doom, PEs are pondering their playbooks, and journalists are jabbering about whether AI kills SaaS, and don't get me started about Private Credit whatever the heck that is.

Tired of the hype, I set out to document reality. A study. Create as many working applications as I could in a set period of time. In the end, I burned out in 75 days. I documented everything. And, the apps are real, 16 are posted in the Apple App Store. Source code published on GitHub, that means the actual code is free. And some of the apps are complex. I rebuilt a digital banking platform, and frankly it's the best one I've built (in years hahaha). The outcomes are real. Software development has changed forever.

This study specifically documents the 75-day effort. It is a pure statement about software abundance and an explosion of the economics around the business of software. It is a study about a very optimistic future filled with more software and filled with more software orchestrators and builders. The amount of software in the world is exploding. The barriers to building software quickly and inexpensively have fallen and fallen quickly. Quality software can be created to solve complex problems. And anyone can build, anyone can become an App Builder. People with domain expertise and a prompt can now build software virtually for free.

Metric	Legacy Hand-Coder	The Software Factory (Abundance)	The Multiplier
Total Effort	4,050 Days (15.5 Years)	**75 Days**	**54x Faster**
Code Produced	405,000 LOC	**405,000 LOC**	N/A
Production Cost	$1,166,400 (Labor only)	**$450** (LLM costs)	**2,592x Cheaper**
Delivery Cost	$1,166,400	**$46,500** (Total Project)	**25x Cheaper**

* The scarcity vs abundance gap

One major point. The advances in building software with AI should not be construed as a commentary on AI's ability to have impact in other areas. In fact, AI is proving to be narrow in its focus of excellence…. This study shows that AI is great at building apps but *narrower than AI boosters claim and more real than skeptics realize.*

The data is hard to put down, read on to learn more, here are a few key stats:

- Humans Drive. On every app the human contribution exceeds 50%.
- Over the 75 days, 310 hours were dedicated to building the apps
- 61% of app build effort is dedicated to requirements, design and prototyping,
- 50% of the LLM effort is planning vs actual writing code
- The daily coding rate for the continuous 75 days was 5,400 lines of code per day
- The overall cost of the experiment was $450 in LLM coding costs and $46,000 in people costs for a total project cost of $46,500.

Remember too, this is a repeatable process. The software factory will scale. The future will be filled with 1,000 people running factories. The software reformation has begun!

Study Background

Defining A Software Factory

This study was completed by a person, a software factory (a piece of software) and an LLM coder engineer. The software factory component requires further explanation. A software factory is a piece of software used by a person to build apps. The software factory toolkit enables a person to build products or applications using a defined, deterministic process. The process is based on Protocol Driven Development (PDD), a requirements-driven software methodology. This is more than a tool; it is a full software production system.

The software factory standardizes the process of building applications and tracks diligently the overall process and software output. Starting with research and requirements, leading into design and prototyping, the emphasis of the process is on up front thinking and final validation of results. This is a concept not new to software development as we have always known, the better our up-front work, the better our output and the less change we had later in the process and that we must validate everything at the end.

Software Factory Test Overview

33 apps were created during this analysis. For 75 continuous days, between January 9, 2026, and March 24, 2026, an experiment was run to see how much software could be created by a single person using a software factory and an LLM engineering team.

During the 75 days, approximately 310 total hours were dedicated to building the applications.

Product development was tracked using an internal tool which recorded progress for all the applications. A standard build stage process was applied to all projects. The stages for the build process were as follows:

	Build Stage	% of Effort
1	Requirements Gathering	51%
2	Design Definition	5%
3	Prototyping	5%
4	Initial Coding	24%
5	App Polish Coding	8%
6	Final Coding	3%
7	App Store Posting (when applicable)	4%

One Person Factory

All roles in the PDD development process were handled by a single person with the assistance of an LLM. The roles broke down as follows:

ROLE	DESCRIPTION
Product Requirements	Background research and requirement documents were prepared for every project.
Product Design	A design brief was prepared for every project including a brief on developing a logo, and design assets for the app build.

Asset Design	Design assets were built by an LLM. This included a logo for all apps and an occasional additional graphic.
Product Prototyping	Prototypes are built as part of the requirements gathering process. The LLM is used to create an HTML prototype and this amounts to 20% of the overall prototyping effort.
Code Assembly/Engineering	All coding was done by an LLM, in most cases Claude Code Desktop.
Language Translation – Full Product	An LLM completed the translation for apps in 22+ languages
Quality Control – QA	An overall AQAP quality audit is run on all apps. Automated regression tests were implemented for 75% of the app code base. App quality tests are run and fully automated. Other QA was performed manually.
Deployment	Deployment was primarily managing the Apple App Store posting process and deployment of the code to GitHub and the Vibin Labs website.

Success Tracking – Lines Of Code

Tracking lines of code written by an engineer or an LLM is not a definitive measurement of output. There are lines of code…. And then, there are lines of code!!! Either way, it is used as a measurement in this study, it was at minimum a way of comparing one app to another in the stable of 33 products. In addition, the speculation of how many lines of code a hand-coder can write, is also a topic of debate. For this paper, we erred on the conservative side of the estimates.

Tech Environment

The applications were built primarily on the Apple platform. Apps were built across multiple devices including iPhones, iPads and desktop. A few of the apps were replicated for the Android platform and the PC desktop platform.

Claude Code Desktop was the primary LLM engineering partner running the LLM side of the study. Claude Code Desktop has a unique ability to manipulate files locally on the desktop computer. This means Claude Code Desktop can complete all the necessary tasks of getting code compiled and running.

The primary coding environment was Xcode for Apple, C# for PC desktop and Kotlin for Android.

Git was used during the development process to track changes to the source code and several of the primary apps are published in a public git repository for all to use freely.

Key Learnings and Data Points

61% Of Build Effort Is Human – Augmented Not Replaced

The big finding!!! People are here to stay. This is a tale of augmentation…. In the one domain where AI has shown the greatest capability, people do 61% of the work. People are doing 61% and then being augmented by LLM engineering. The people part includes research, requirements, design, and prototyping. The person becomes augmented and can produce at very high rates. The AI "worker" does about 35% of the effort.

Across the 33 applications of varying complexity, human requirements effort ranged from 45% to 75% of total effort, with a portfolio average of 61%. In all cases the person did more than half the work. Not in a single instance did the AI effort exceed the human effort, not once, not a single app. People are not being replaced. The process is impossible to complete without them. What changes is the rate of output. Now, one person can produce the results of an entire team in a fraction of the time and at virtually no cost.

The Cost Was $0.000987 Per Line Of Code - Virtually Zero

33 products were built with the PDD software factory process representing 405,000 lines of code. The LLM coding costs in isolation were $450. That equates to $0.000987 per line of code. Code is now virtually free to produce.

There were costs associated with the person orchestrating the factory, but those exist whether requirements are handed over to an LLM or a hand coder. So, in isolation the LLM coding costs can be compared to a hand coder. Any way you do the math, it doesn't look good for the hand-coding movement.

1 App Every 2 Days – Products Were Built Very Fast

33 apps across 63 platforms representing 405,000 lines of code were built in 75 days, that's a lot of output by any measure. The average time for a full build of a product was 9 days. Toward the end of the test, the average was closer to 2-3 days.

The overall process of building the apps took a total of 310 hours. Approximately 100 hours was dedicated to the LLM writing code for the 33 applications.

- 5,400 lines of code per day for 75 days
- 4,050 lines of code per hour for the 100 hours of LLM coding
- 1,306 lines of code per hour for the 310 total build time

As an interesting note, approximately 50% of the LLM time is spent planning, Claude Code spends 50% of its time doing analysis and building detailed plans and 50% writing code.

61% Of Build Time On Thinking – Research, Requirements, Design & Prototyping

On average, 61% of the effort is thinking, 35% is coding. And the benefit of this up-front thinking is clear. When apps have well defined requirements, the overall process goes much quicker and there is less iteration.

- Apps with well-defined requirement were 70% complete in the first build and took 30% of the time to iterate to complete the product.
- Apps with less upfront requirements had longer build times of 15+ days and had much lower ratios, usually closer to 50/50 in terms of the build to iteration split.

The PDD process benefits from a requirements-driven approach.

Here is the overall distribution of effort by task:

	Build Stage	% of Effort	
1	Requirements Gathering	51%	Thinking
2	Design Definition	5%	
3	Prototyping	5%	
4	Initial Coding	24%	Coding
5	App Polish Coding	8%	
6	Final Coding	3%	
7	App Store Posting (when applicable)	4%	Admin

Setting up app and product distribution, i.e., the App Store, can be a real…. Challenge! On some apps, like the apps that support 20+ languages, a multilanguage App Store posting can take 1-2 hours, and do not get me started about posting 20 sets of localized screen shots. 22 languages, 2 platforms, 5-6 screens per…. Yes, that's hundreds of screen shots, and yes…. I built software called Angie to manage that process too…

Better Documented Apps = 70% Faster Build

The other clear benefit of the requirements-driven PDD process is speed. Apps with clear upfront requirements were built 70% faster and with lower iteration as described above. No products created in less than 3 days had an initial build to iteration ratio of lower than 70/30, 70% of the app was built correctly in the first instance. Whereas apps that needed more iterations were usually lacking detailed requirements and a true vision for the app when started.

The factory also got faster. It took 20 days to write the first 100,000 lines of code and 9 days to write the last 100,000 lines of code. The process improved, the apps got bigger and more complex and the overall build times decreased as the factory started reusing components and finding leverage in the factory model.

Milestone Reached	Date
0 LOC	Jan 9
10,000 LOC	Jan 18
50,000 LOC	Jan 25
100,000 LOC	Jan 29
200,000 LOC	Feb 12
300,000 LOC	Feb 28

400,000 LOC	Mar 9

Mini Case Study On Vibe Efficiency - Requirements To Code In 3 Hours

I always wanted a fully automated test suite for regression testing. That can be almost impossible to build on a legacy system as first and foremost not many of us built it that way to begin with so test automation is bolted on. Well, not this time. Full regression test it would be.

The digital banking platform and my software factory platform received full testing automation.

- From gathering requirements to full implementation, it took 2.5 hours for digital banking and about 1 hour for Augie software factory as I reused the testing framework for the second build.
- The digital banking platform has 26 test suites and 373 individual tests
- The Augie software factory tool as 19 test suites and 287 individual tests.
- Both digital banking and Augie have an additional 50+ point application quality audit (AQAP) which is also automated and run after each change to a feature or the platform. The AQAP ensures adherence to high code quality standards.
- 100% coverage. The tests touch every part of the transaction flow; no function or feature is left out. This includes as much testing of back-end connections as is possible.
- 100% Test-Structured Code. The code has been structured to support automated testing. The code was written in such a way to make testing possible and very efficient. This was done as the code was being built, purpose-built source code to support an automated environment
- The tests run in 5-10 minutes maximum. Which means even the smallest change gets a full regression test. It also means you can run the test multiple times a day as an additional level of production support, another option that is almost impossible with legacy code.
- Test Dashboard and 10+ Detailed Reports. The tests are fully instrumented on screen in the admin console called Oppy. Detailed and auditable reports are created for future inspection.

In record time and with minimal effort major components needed to deliver in a highly regulated environment, are not only possible but easy to achieve. Software produced by the software factory has quality built in and at speed.

33 Production Ready Apps

The software factory produced apps with a repeatable standard process. 33 apps were produced during the study. Sixteen (16) of the apps are in the Apple App Store. The apps ranged from simple personal utilities like a pedometer to track steps to more sophisticated banking applications. A full list is provided below.

- All the apps were native device applications. There were three (3) HTML websites.
- SWIFT was used as the primary language for app development

- Xcode was the development tool used to build the apps.
- Kotlin and C# were also used for Android and PC desktop.

63 Device Specific Apps Represented

Most products were built for more than one device. There were in fact 63 separate app targets across the 33 apps. A few apps were single platform, such as a few desktop apps, but for most, the apps were built for iPad, iPhone and Desktop. In a few instances the products had a server implementation in addition to front end applications.

A factory protocol was developed to manage the process of building apps across platforms; the App Conversion Protocol (ACP). Using the ACP, all apps were built across platform:

- SWIFT apps can now share 90% of the same code base. iPhone and iPad apps are basically developed for the same cost.
- Desktop can be added with about 10% additional effort thinking through screen layout and needing to occasionally build device specific code.

Seven Products Used Artificial Intelligence

Seven (7) of the products used artificial intelligence within the app. Some had direct API access and some integrated AI within embedded browsers. AI was used for research, content preparation and guided conversations.

The other important element of LLM usage was what is referred to as intent routing. The LLM is unique in its ability to establish explicit intent from a text string. That is a handy tool when building any kind of conversational interface. Latency for using the LLM to establish intent can range from 3 seconds to 5 seconds. That does make for a choppy conversation but can be used effectively in select user engagement situations.

Multi Language Products Are Normalized

Multilingual apps is one of the easier outcomes coming from the software factory approach. 20% of the 33 apps were developed to support 22+ languages.

Multi-language apps have 65% of their downloads coming from outside the USA.

It takes the factory approximately 30-45 minutes to make an app multi-language as there is a standard protocol for converting apps to multi-lingual easily applied to each app slated for conversion. Standard processes like this thrive in a software factory setting. For a few hours of software factory effort, a Builder can pick up 65% more downloads, that's just a good business decision.

Case Study – Digital Banking Platform

One of the products delivered during the test was a fully functioning digital banking platform. This tested if a software factory and PDD could produce a complex application. Here are the basic stats for the digital banking platform:

- 4 Devices - 1 Desktop App, 1 iPhone App, 1 Android App, 1 Web App
- 60 hours build time (72% requirements and 28% coding effort)
- 135,000 Lines of Code
- 40 Consumer and Business Modules
- 65+ Feature On/Off Switched Driven by a Full Entitlements Engine
- 50+ Automated Operational Reports - Oppy Console
- 373 Automated Test Regression Suite Running in 5 Minutes
- 100% Open Banking API Feature Coverage
- 3-Tiered Full Server Architecture
- 50+ Point Application Quality Audit Protocol - Testing & Audit (AQAP)

The software factory process created a complex, enterprise grade application with a graceful module architecture. The system has enhanced security over and above a typical digital platform today, is fully compliant, scalable and ready for production. More features, better infrastructure, full regression test suite and full operational instrumentation, delivered at near zero cost.

The Digital Platform Build: 135,000 LOC 60 Hours				
	LOC per Hour	**Effort In Hours**	**Labor Cost**	**Total Costs**
Hand Coding	12-15 LOC per hour 100 LOC per day	1,350 days x 8 Hrs 10,800 hours	$75/hour	$810,000
Software Factory	2,250 LOC per hour	60	$148/hour	$8,880

* *Even when granting the Legacy model a generous output of **100 LOC/day** (nearly double the industry average for fully tested enterprise code) and a budget labor rate of **$75/hr**, the Software Factory remains **91x more cost-effective** and delivers the product **180x faster.***

Case Study – Agency Tracking App

An advertising agency rebuilt an internal app used to track customer results. The original web app was replaced with a full desktop implementation for Mac and PC with a hosted server element. The app is currently in testing.

- 3 Devices – Server, Mac and PC Desktop
- 15 hours of build time (70% requirements and 30% coding)
- 20,000 Lines of Code
- Multiuser app with full cloud-based server component
- LLM integration for article URL extraction and preparation

Using traditional methods of development, this app would have taken months to build, and the cost would never have been justified, the business would not have replaced the old system. Using a software factory, this Agency has delivered a new app that is 50%

more efficient for the employees and delivers 100% more new features and functions. All for close to zero cost. This is the software reformation at work.

The Agency Build: 20,000 LOC 15 Hours				
	LOC per Hour	**Effort In Hours**	**Labor Cost**	**Total Costs**
Hand Coding	12-15 LOC per hour 100 LOC per day	200 days x 8 Hrs 1,600 hours	$75/hour	$120,000
Software Factory	2,250 LOC per hour	15	$148/hour	$2,200

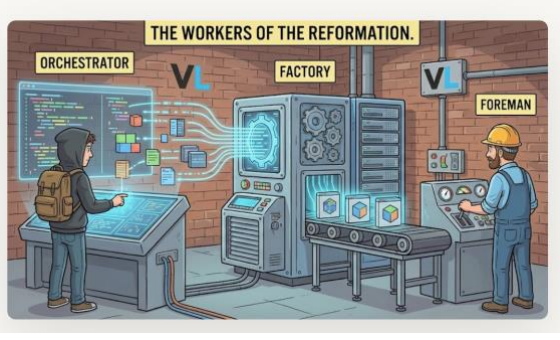
Software Industry Implications

The factory test documents one case study demonstrating how fast and at what cost we can now produce software. What does this mean for the software industry?

- This study demonstrated that production costs dropped to $0.000987 per line of code while traditional methods remain 25x more expensive, will vendor economics which rely on software scarcity face pressure? Will some business decide to build software vs buying it at these new economics?

- A single domain expert built a large complex product in 50-60 hours; how will the legacy vendors respond given their large development organizations and current cost structures?

- The test shows how the new model relies on 61% of the work being done by business domain experts, how will the mix of people change at the legacy vendors? The new process demands more experts.



And there is now evidence this is happening; companies are vibe coding!!! A recent Wall Street Journal article (Lin, Belle. 'Companies Aren't Ripping Out Business Software

for AI. Here's What They're Doing Instead.' March 23, 2026) cited 4 examples where major enterprises are using vibe-coding techniques to replace software vendors. Some use AI to build new software, some use AI agents to replace existing processes. All are examples of what the future might look like.

The operational implications of software abundance including support, maintenance, and how we will operate and keep the software running represent a significant consideration that extends beyond the scope of this production-focused study. Fortunately, there are answers to these challenges, the topic to be explored under a separate report.

Next Exciting Steps

This is one test, one study conducted by one person. I am not an engineer and yet, I created 33 viable products, from simple to complex. The apps are in the App Store and posted to Git. The digital bank platform exists.

Now everyone else needs to do their own test. Build and create apps, let the orchestrators loose and let's watch the results. Go do it yourself. Run your own factory!

And imagine if you will…. This idea scales. It replicates. Thousands, maybe someday many more will be creating apps and software will proliferate like never before. If one non-engineer can build 33 apps in 75 days, what can 1,000 people accomplish? Let's find out.

The software factory experiment produced working products. And humans are in charge. The software factory is at their disposal. The conclusion is narrower than the boosters claim and more real than skeptics realize!

Go Code!!! Go run your own software factory!!!

App Name	Stage	Platforms	Start Date	Build # Days	Current LOC	Files	Active Days	Effort Hours	LOC/Hour
Angie Factory Store	App Live	Mac	2026-02-02	8	25,055	32	40	27.1	924.5
Augie Digital Banking Platform	Coding	iPhone,iPad,Mac, Server		11	130,032	773	35	61.5	2,114.3
Augie EDU	App Live	Mac	2026-02-05	2	7,953	26	37	4.0	1,988.2
Augie EDU Data Content Admin Console	App Live	Mac	2026-02-05	1	614	6	37	2.6	236.2
Augie Pay	Final Coding	iPhone,iPad,Mac	2026-01-28	10	18,095	42	46	9.4	1,925.0
Augie Software Factory	App Live	Mac	2026-01-08	29	32,033	97	66	34.0	942.1
Augie V 2.0	App Live	Mac,Server	2026-03-15	1	30,636	67	39	11.4	2,687.4
Cloudless CRM	Coding	iPhone,iPad,Mac	2026-03-22	1	8,369	17	4	3.7	2,292.9
Cloudless Decipher	App Live	iPhone,iPad	2026-01-30	8	5,871	44	44	9.0	653.8
Cloudless Documents	App Live	iPad,Mac	2026-03-12	1	9,219	22	11	6.8	1,349.8
Cloudless Logger	App Live	Mac	2026-02-08	1	6,031	36	35	3.7	1,630.0
Cloudless Notes	App Live	iPhone,iPad	2026-02-11	1	7,999	22	32	9.0	893.7
Cloudless Pay	R&D	iPhone,iPad,Mac	2026-03-23	1	4,637	11	3	3.9	1,189.0
Cloudless Photo	App Live	iPhone,iPad	2026-01-24	50	6,614	39	12	6.1	1,084.3
Cloudless Recipes	App Live	iPhone,iPad,Mac	2026-03-09	4	13,643	82	14	9.0	1,515.9
Cloudless Reminders	App Live	iPhone,iPad,Mac	2026-01-29	9	3,858	24	45	6.5	593.5
Cloudless Scan	App Live	iPhone,iPad	2026-01-22	16	4,552	45	51	9.7	471.7
Cloudless Steps	App Live	iPhone	2026-01-22	15	6,326	53	52	6.3	999.4
Cloudless Symptoms	Final Coding	iPhone,iPad	2026-01-31	7	3,070	85	42	4.9	626.5
Cloudless Time	App Complete	iPhone,iPad,Mac	2026-03-16	4	7,263	29	8	3.9	1,886.5
cloudless Travel	Final Coding	iPhone,iPad,Mac	2026-03-18	2	11,236	45	6	6.8	1,652.4
Cloudless Watermark Maker	App Live	iPhone,iPad,Mac	2026-02-14	2	7,649	30	30	8.8	869.2
Conversational Bank Tester	R&D	iPhone	2026-01-31	43	1,613	11	8	1.0	1,613.0

DailyNet-Dev	Final Coding	iPhone,iPad,Mac	2026-01-28	10	170	30	40	6.2	27.6
Lumi Logger 1.0	App Live	Mac,Server	2026-03-18	8	0	14	4	5.2	0.0
My Next Show	App Live	iPhone	1/9/26	29	4,737	51	64	6.7	711.3
Software Reformation Website	App Live	Other	2026-02-27	1	1,981	18	36	1.3	1,489.5
Sporting Wire	App Live	iPhone	2026-01-26	11	4,843	21	47	3.3	1,476.5
Subscription Killer	App Live	iPhone,iPad	2026-01-15	23	13,074	677	58	10.8	1,212.8
Vibin Labs Website	App Live	Other	2026-02-15	1	3,567	46	37	4.3	823.8
Wacky Weather	App Live	iPhone	2026-01-25	13	4,531	30	49	7.0	643.6
Wheel Barrow Industries Website	App Live	Android	2026-02-15	1	2,607	30	33	1.3	1,960.2
PR Firm Customer Reporting	Coding	Mac	2026-02-26	2	18,987	27	23	12.0	1,582.2

App Effort Build Breakdown

App Name	Build Type	Build Name	Requirements	Design	Prototyping	Initial Coding	Finish Coding	App Store Polish	Store Posting	Total Hours
Angie Factory Store	initial	initial build	2.00	1.00	1.00	1.00	0.50			5.50
Angie Factory Store	update	Update 1	13.50		0.20	5.50	2.20	0.20		21.60
AF TOTAL			15.50	1.00	1.20	6.50	2.70	0.20	0.00	27.10
Augie Digital Banking Platform	initial	initial build	4.00	0.50	0.50	2.00	1.00			8.00
Augie Digital Banking Platform	update	Update 1	31.00	4.00	4.00	9.50	5.00			53.50
Augie EDU	initial	Initial Build	1.50	0.50		1.00	1.00			4.00
Augie EDU Data Content Admin Console	initial	Initial Build	1.00	0.25	0.25	1.00	0.10			2.60

Augie Pay	initial	Initial Build	2.50	0.20	0.50	1.00	0.50			4.70
Augie Pay	update	Update 1	3.00		0.20	1.00	0.50			4.70
Augie Software Factory	initial	Initial Build	10.00	2.00	1.00	7.00				20.00
Augie Software Factory	update	Update 1	7.00	1.50	0.50	5.00				14.00
Augie V 2.0	initial	Initial Build	2.50	0.20		1.00	0.50	0.50		4.70
Augie V 2.0	update	Update 1	3.00	0.20		2.00	1.00	0.50		6.70
Cloudless CRM	initial	Initial Build	1.50	0.20	0.20	1.00	0.50	0.25		3.65
Cloudless Decipher	initial	Initial Build	2.00	0.20	0.20	1.00	0.20	1.00	0.30	4.90
Cloudless Decipher	update	Update 1	2.00			1.00	0.50	0.25	0.33	4.08
Cloudless Documents	initial	Initial Build	3.50	0.50	0.25	1.75	0.25	0.25	0.33	6.83
Cloudless Logger	initial	Initial Build	0.75	0.25	0.25	0.50	0.20			1.95
Cloudless Logger	update	update	1.00			0.50	0.25			1.75
Cloudless Notes	initial	Initial Build	1.50	0.20	0.20	0.80	0.50	0.50	1.00	4.70
Cloudless Notes	update	Update 1	2.00			1.00	0.25		1.00	4.25
Cloudless Pay	initial	Initial Build	2.00	0.10	0.10	1.00	0.50	0.20		3.90
Cloudless Photo	initial	Initial Build	2.50	0.20	0.20	1.50	0.50	0.20	1.00	6.10
Cloudless Recipes	initial	Initial Build	4.00	0.50	0.20	2.00	1.00	1.00	0.30	9.00
Cloudless Reminders	update	Update 1	2.00			0.80	0.50	0.20	1.00	4.50
Cloudless Reminders	initial	Initial Build	1.00	0.25	0.25	0.50				2.00
Cloudless Scan	initial	Initial Build	1.50	0.20	0.20	1.50	0.25	0.25	1.00	4.90
Cloudless Scan	update	update 1	2.00			1.00	0.50	0.25	1.00	4.75
Cloudless Steps	update	Update 1	1.50			1.00	0.25	0.10	1.00	3.85
Cloudless Steps	initial	Initial Build	1.00	0.25	0.25	0.50	0.15		0.33	2.48
Cloudless Symptoms	initial	Initial Build	3.00	0.20	0.20	1.00	0.50			4.90

Cloudless Time	initial	Initial Build	1.00	0.20	0.20	1.00	0.25	0.20	1.00	3.85
cloudless Travel	initial	Initial Build	3.00	0.50	0.30	1.50	1.00	0.50		6.80
Cloudless Watermark Maker	initial	Initial Build	4.00	0.50	0.50	2.00	1.00	0.50	0.30	8.80
Conversational Bank Tester	initial	Initial Build	0.67			0.33				1.00
DailyNet-Dev	initial	Initial Build	3.00	0.20	0.20	1.75	1.00			6.15
Lumi Logger 1.0	initial	Initial Build	3.00	0.20	0.20	1.00	0.75			5.15
My Next Show	initial	Initial Build	1.50	0.25	0.25	0.75	0.25		0.33	3.33
My Next Show	update	Update 1	2.00			1.00			0.33	3.33
Software Reformation Website	initial	Initial Build	1.00			0.33				1.33
Sporting Wire	initial	Initial Build	2.00	0.10	0.10	0.83			0.25	3.28
Subscription Killer	initial	Initial Build	3.50	0.20	0.20	1.50	0.75	0.25	0.63	7.03
Subscription Killer	update	update 22 languages	1.00			1.00	0.50	0.25	1.00	3.75
Vibin Labs Website	initial	Initial Build	1.00			0.33				1.33
Vibin Labs Website	update	Update 1	2.00			1.00				3.00
Wacky Weather	update	Update 1	2.00			1.00	0.33	0.15	0.33	3.81
Wacky Weather	initial	Initial Build	1.50	0.25	0.25	0.75	0.15		0.33	3.23
Wheel Barrow Industries Website	initial	Initial Build	1.00			0.33				1.33
PR - Customer Reporting	initial	Initial Build	5.00	0.50	0.50	2.50	1.25			7.75
PR- Customer Reporting	update	Update 1	3.00			2.00	0.75			4.25