UPPSALA
UNIVERSITET

A comprehensive summary and categorization of physical quantity libraries

Oscar Bennich-Björkman

Subject: Information Systems
Corresponds to: 30hp

Presented: Spring 2018
Supervisor: Steve McKeever
Examiner: Görkem Pacaci

Department of Informatics and Media

Abstract

In scientific applications, physical quantities and units of measurement are used regularly. If the inherent incompatibility between these different units is not handled properly it can lead to major, and sometimes catastrophic, problems. Although the risk of a miscalculation is high and the cost equally so, almost no programming languages has support for physical quantities. Instead developers often rely on external libraries to help them spot these mistakes or prevent them all together.

There are several hundred of these types of libraries, spread across multiple sites and with no simple way to get an overview. No one has summarized what has and has not been achieved so far in the area leading to many developers trying to 'reinvent the wheel' instead of building on what has already been done. This shows a clear need for this type of research.

Employing a systematic approach to look through and analyze all available physical quantity libraries, the search results were condensed into 82 libraries which are presented in this thesis. These are the most comprehensive and well-developed, open-source libraries, chosen from approximately 3700 search results across seven repository hosting sites. In this group, 30 different programming languages are represented. The goal is for the results of this thesis to contribute to a shared foundation on which to build future libraries as well as provide an easy way of spreading knowledge about which libraries exist in the area, thus making it easier for more people to use them.

Keywords

Physical quantities, libraries, summary, overview, units of measurement, open-source

Sammanfattning

I vetenskapliga program används fysikaliska storheter och deras respektive enheter (meter, sekunder, kilogram) regelbundet. Om man försöker addera eller jämföra inkompatibla enheter i dessa program kan det leda till stora, ibland katastrofala, problem. Trots att risken för misstag är stor och kostnaden likaså så stödjer nästan inget programmeringsspråk användandet av storheter. Istället måste utvecklare förlita sig på externa bibliotek för att upptäcka dessa misstag elle förebygga dem helt.

Det finns hundratals av dessa bibliotek, spridda över många webbsidor, utan något enkelt sätt att få en överblick. Ingen har sammanfattad vad som åstadkommits i området hittills vilket har lett till att många utvecklare försöker 'återuppfinna hjulet' istället för att vidareutveckla det som redan blivit gjort, vilket visar på ett tydligt behov av den här sortens forskning.

Efter att ha analyserat alla tillgängliga fysikaliska storhets-bibliotek på ett systematisk vis presenteras i den här uppsatsen 82 av dessa. De är dem mest omfattande och välutvecklade open-source bibliotek som blivit valda från omkring 3700 sökresultat över sju olika sidor. Trettio olika programmeringsspråk finns representerade i den här gruppen. Målet är att resultaten från denna uppsats ska bidra till en delad grund på vilken framtida bibliotek kan byggas och tillhandahålla ett enkelt sätt att sprida kunskap om vilka bibliotek som existerar inom detta område och på så sätt göra det enklare för fler personer att använda dem.

Table of contents

List of figures

List of tables

1 Introduction

1.1 Background

On the morning of September 23rd, 1999, NASA lost contact with the Mars Climate Orbiter, a space probe sent up a year prior with the mission to survey Mars. The probe had malfunctioned, causing it to disintegrate in the upper atmosphere. A later investigation found that the root cause of the crash was the incorrect usage of Imperial units in the probe's software (NASA, 1999). This seemingly trivial mistake ended up costing more than 300 million dollars and several years of work ("Mars Climate Orbiter Fact Sheet", n.d.).

There are several other examples of situations similar to that which lead to the Orbiter crash, such as:

- In 1983, a Boeing 767 ran out of fuel an hour into its flight as a result of a unit miscalculation when pumping fuel. Fortunately, the pilot (who was an experienced glider) managed to land the airplane at an airfield close to a town called Gimli and only two passengers suffered some minor injuries. Because of this, the plane became known as the "Gimli Glider". (Witkin, 1983) ("Unit Mixups", 2009).

- In 1984, the space shuttle "Discovery STS-18" accidentally ended up being positioned upside down because the engineers had mistaken feet for miles (Neumann, 1992).

- In 1999 Korean Air Flight 6316 crashed as a result of a misinterpretation of feet and meters. Five people were killed and 37 injured ("Unit Mixups", 2009) (Ranter, 2018).

- In 2003 a roller coaster ride in Tokyo broke down because of an error arising from the wrongful use of metric units in the design papers for the ride ("Unit Mixups", 2009).

These examples make it clear that these types of situations are both prevalent and costly, not only financially, but also costing time and in some cases even lives. How can the mistakes that cause them be prevented? One way is through the use of physical quantity libraries.

1.2 Problem description

What the examples in the previous section have in common is that they involve human error in relation to units of measurement (metres, seconds, kilograms...) used in software systems. It can be both mistakes made *within one unit system* (as in the case of Discovery STS-18) or errors stemming from wrongful conversions *between different unit systems* (what happened with the Mars Climate Orbiter or Flight 6316).

To better understand the underlying problem, consider the following scenario. There are two programmers working on a system that manages physical quantities[1] using a popular language such as C#, Java, or Python. The first programmer wants to create two quantities that will be used by the second programmer at a later stage. Because the language does not have support for this type of construct, he or she decides to do it using integers and adding comments, like this:

```
int mass = 10; // in tonnes
int acceleration = 10; // in m/s
```

Now the second programmer wants to use these values to calculate force using the well-known equation $F = m * a$:

```
int force = mass * acceleration; // 100 N
```

The variable *force* will now have the value of 100, assumed to be 100 N. The issue is that the variable *mass* is actually representing 10 tonnes, not 10 kilograms. This means that the actual value of the force should be 100000 N (instead of 100 N), off by a factor of one thousand. Because the quantities in this example are represented using integers there is no way for the compiler to know this information and therefore it is up to the programmers themselves to keep track of it.

If the second programmer would have seen the comments that specified this the problem could have been avoided, but imagine how easy it would be to make this kind of mistake if the program contained thousands of variables and complex equations. This is in essence the fundamental problem that can lead to the catastrophic events that were described previously.

The only reliable way to solve this is try to remove the human element by having automatic ways to check calculations and make sure they are correct. This type of automatic check is potentially something that could be done at compile time in a

[1] See Section 2.1 for an explanation of how physical quantities relates to units of measurement and unit systems.

strongly typed language, but unfortunately very few languages have support for units of measurement (see Section 2.2). Instead, it is up to the software developers to create these checks themselves.

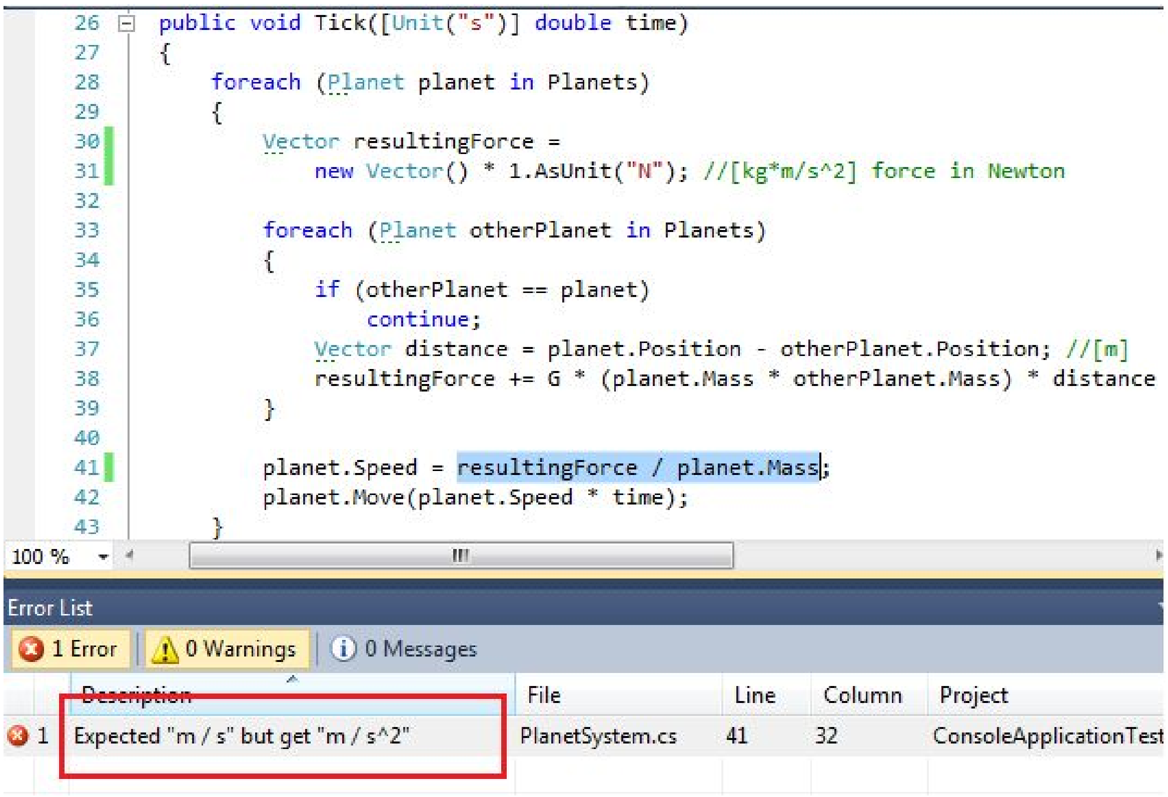
One example of how this can be done is to make sure the compiler knows what quantities are being used by encapsulating this into a class hierarchy, with each unit having its own class. The scenario above would then look like this instead:

```
Tonne mass = 10;
Acceleration acceleration = 10; // in m/s

Force force = mass * acceleration; // 100000 N
```

Compared to the previous example, here the compiler now knows exactly what it is dealing with and thus the information that the mass is in tonnes is kept intact and the correct force can be calculated in the end.

This type of solution not only means that differences in magnitude and simple conversions are taken care of but also that any (or most) erroneous units being used in an equation can be caught at compile time. An example of this can be seen in Figure 1.



```
26  public void Tick([Unit("s")] double time)
27  {
28      foreach (Planet planet in Planets)
29      {
30          Vector resultingForce =
31              new Vector() * 1.AsUnit("N"); //[kg*m/s^2] force in Newton
32
33          foreach (Planet otherPlanet in Planets)
34          {
35              if (otherPlanet == planet)
36                  continue;
37              Vector distance = planet.Position - otherPlanet.Position; //[m]
38              resultingForce += G * (planet.Mass * otherPlanet.Mass) * distance
39          }
40
41          planet.Speed = resultingForce / planet.Mass;
42          planet.Move(planet.Speed * time);
43      }
```

Error List

1 Error | 0 Warnings | 0 Messages

	Description	File	Line	Column	Project
1	Expected "m / s" but get "m / s^2"	PlanetSystem.cs	41	32	ConsoleApplicationTest

Fig 1. Compilation error example (Dieterichs, 2012).

In the example shown in the picture above, the programmer makes a mistake when trying to calculate speed. Instead of using the correct units (which is length per time, m/s) he or she uses meters divided by seconds squared (m/s^2). Luckily, this gets caught by the compiler and a compilation error is thrown informing the programmer that the wrong unit was used.

Another way this problem can manifest itself is when a quantity is expressed in the correct unit but with the wrong magnitude. This type of problem is similar but distinctly different as it is not incorrect from a physical perspective but from a human one. For example a programmer could construct a unit, *Age*, which represents a person's age:

```
    Age personAge = 1000; // in years
```

Compared to before, there is nothing physically wrong with this value, and it would be able to be used without issues in any equation. But at the same time we know that 1000 years is an unreasonable number for the age of a person. A real-world example of this type of scenario happened when the Norwegian team accidently ordered 15000 eggs instead of 1500 during the 2018 Winter Olympics. This occurred because in the South Korean counting system the difference between these two numbers is only one syllable, making it an easy mistake (Livsey, 2018).

One way to prevent this is to introduce the idea of valid 'ranges' for units. For example, the max value for the unit 'Age' could be set to to 150 years and the max number of eggs to 2000. The same scenario would then result in an error, similar to the one shown in Figure 1, as 1000 years and 15000 eggs would not be inside the legal range for each respective object.

The major issue with developers having to implement the types of solutions described here is that considerable time and effort are required to do so. Making a class hierarchy similar to the one illustrated above could potentially involve hundreds of units and thousands of conversions and one mistake can be one too many. Because of this it is not always possible to do and even so, it might not be prioritized as the consequences of making these mistakes can easily be underestimated.

One way to tackle this issue and be able to safely use physical quantities without spending precious development resources is to use what others have already done, in the form of free, open-source, software libraries. These libraries can provide the class-structure and logic that is needed and can (often) be easily integrated into an already existing code base. These types of libraries do already exist, several hundreds in fact. The problem thus is not the lack of these solutions but the opposite, that they are so numerous that it is hard to get an overview. Moreover, when looking deeper, it quickly becomes apparent that there is no agreed upon standard in the area and a general lack of cooperation. This results in most library developers creating their own versions from scratch without referencing the work of others. Even though they all try to solve the same fundamental problem, the libraries are developed in isolation. In other words, they try to reinvent the wheel.

1.3 Research goal

The main goal of this thesis is to provide a summary and categorization of all libraries (specifically open-source) that handle the use of physical quantities through a comprehensive and systematic approach. This will help solve the problems described in the previous section and on such a scale, this type of effort has not yet been undertaken, even though there is a clear need for it (see Section 2.3 for a thorough look at related work).

The categorization of the libraries is both done in terms of quality, only the best ones being presented, as well as categorizing these selected libraries according to how they work to solve the problem at hand. This latter type of categorization should be seen as secondary to the first, and is employed mainly as a way of giving further information about this group of libraries.

The results from this study are mainly aimed at two groups. The first group are developers who are in need of a library that handles physical quantities in their preferred language but do not have the time or resources to find and analyze all relevant alternatives themselves. Using the results from this thesis to choose a library can help them save considerable time and effort. The second group are people who wish to develop a unit library of their own or contribute to an existing one. In this case, the results can help them 'stand on the shoulders of giants', providing them with a collection of information about what has already been undertaken, categorized and summarized in an easily accessible way.

These goals are summarized into two main research questions:

1. *What physical quantity libraries exist and which ones are the most comprehensive?*

2. *What different approaches do these libraries employ when dealing with physical quantities. How can these approaches be categorized?*

1.4 Delimitations

There are three primary delimitations that should be addressed here. Firstly, the main results and analysis focuses specifically on libraries and excludes different kinds of tools. These tools include unit calculators, dimensional analysis tools, or other executable files related to the checking of physical quantities. Although these are related to the problem at hand, they cannot be used in the same way as libraries and therefore they have not been included in the main results.

Secondly, the focus of this study is not to analyze the libraries in-depth, but rather provide a comprehensive overview and summary of what is available. The choice to focus on breadth over depth was a prioritization that had to be made as there are far too many libraries to do both types of analysis in this thesis alone.

The final delimitation is that only libraries that are open-source are included. There are two main reasons for this choice. The first reason is because of the need to be able to access the source code of any library that is included in the results, as this is required to accurately determine how the library works. The second reason is that the libraries being open-source means that anyone who, after reading this thesis, would like to utilize one of these can do so immediately without any hesitations nor the need to pay for it. Open-source also means that it is easy for a reader to contribute to one or several of the projects and become an active part of the development of the library if that is something they want to do. With this being said, a cursory analysis seems to indicate that there are very few, if any, libraries that are not open-source, meaning that this delimitation does not necessarily affect the results in any major way.

2 Theory

2.1 Fundamental concepts

As this thesis aims to find and categorize libraries that handle physical quantities, some of the fundamental concepts related to this area of physics will be briefly presented here.

2.1.1 Physical quantities, units, and dimensions

The technical definition of a physical quantity is a “property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference” (Joint Committee for Guides in Metrology, 2012).

To explain this further, each quantity is expressed as a number (the magnitude of the quantity) with an associated unit ("BIPM - quantities and units", 2014). For example you could express the physical quantity of *length* with the unit *metre* and the magnitude 10 (10 m). However, the same length can also be expressed using other units such as centimetres or kilometres, at the same time changing the magnitude (1000 cm or 0.01 km). Keeping the same physical quantity consistent across multiple units is one of the main functions that the libraries presented in this thesis provides, and something that if not kept in check can have major consequences (as described previously).

Physical quantities also come in two types, one called *base quantities* and the other *derived quantities*. The base quantities are the basic building blocks and the derived quantities are built from these. The base quantities and derived quantities together form a way of describing any part of the physical world (Sonin, 2010). For example length (meters) is a base quantity, and so is time (seconds). If you combine these two base quantities you can express velocity (meters/second) which is a derived quantity. The International System of Units (SI) defines seven base quantities (length, mass, time, electric current, thermodynamic temperature, amount of substance, and luminous intensity) as well as a corresponding unit for each quantity ("Essentials of the SI: Base & derived units", n.d.).

These physical quantities are also organized in a system of dimensions, each quantity representing a physical dimension with a corresponding symbol (L for length, M for mass, T for time etc.). Any derived quantity can be defined by a combination of one or several base quantities raised to a certain power. These are called “dimensional exponents” ("BIPM - dimensions", 2014). Dimensional exponents are not a type of unit or quantity in themselves but rather another way to describe an already existing quantity in an abstract way. Using the same example of velocity as before, it can be expressed as $L*T^{-1}$, L representing length and T^{-1} representing this length being divided by a certain time (dividing something that has been raised to the second power would result in the exponent being -2 instead of -1, and so on). Although from a physical perspective each unit can be defined in this way, it is not necessarily the way they are defined in a physical quantity library. Therefore the inclusion or exclusion of dimensional exponents is one potential way of categorizing these libraries (this idea is further described in Section 3.3.1).

As was previously discussed, performing calculations in relation to quantities, units, and dimensions is often complex and can easily lead to mistakes. One way to try to deal with this (outside of unit libraries) is to do what's called a "dimensional analysis". One example of a type of dimensional analysis is to check for dimensional homogeneity, meaning that both sides of an equation has equal dimensions (which they should have) (Sonin, 2001). The concept of dimensional analysis is also relevant to the libraries described in this thesis, as they often employ similar ways to check for errors. There are also specific software tools that can perform dimensional analysis on existing code (for example described by Cmelik & Gehani, 1988).

2.1.2 Unit systems

The examples in the previous section were all based on the "International System of Units" (SI) which is the most used and well known unit system, but there exists several other systems that these physical quantities can be expressed in, each with different units for the same quantity. Other examples include the Imperial system, the Atomic Units system, and the CGS (centimetre, gram, second) system. These have evolved over time and branched off from each other (see Figure 2) (Clarke, 2016).

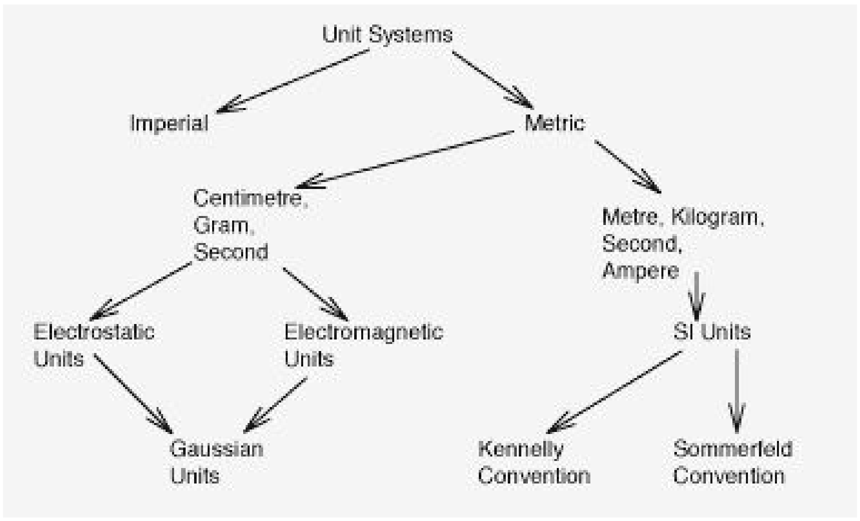


Fig 2. The evolution of unit systems (Clarke, 2016).

To give an example of how these systems relate to units and quantities you can think of how to express a certain length in the SI system, for example 2 metres. 2 metres in the SI system is equivalent to about 6.6 feet in the Imperial system. The same quantity (length) is expressed in both instances but by different units, *metres* in one system and *feet* in the other.

Although the SI system is the most well used, the Imperial system is still being used heavily in the United States, which means that keeping track of quantities expressed in different unit systems is important. Therefore, this is a type of functionality that most quantity libraries implement. In relation to this it is also interesting to note that although the Metric system is not widely used in the United States, since 1988 it has technically been the "preferred system of weights and measures" ("15 U.S. Code § 205b - Declaration of policy", 2018).

2.2 Existing language support

One of the main reasons why third-party libraries are needed to solve the issues related to units of measurement is because there are almost no programming languages where this exists as a construct.

In fact, the only language in the 20 most popular programming languages on the TIOBE index ("TIOBE Index | TIOBE - The Software Quality Company", 2018) that support units of measurement is Apple's Swift language ("NSMeasurement - Foundation | Apple Developer Documentation", n.d.). The only other well-known language to support units of measurement at the moment is F# ("Units of Measure in F#: Part One, Introducing Units", 2008). Although compared to Swift, F# is in the bottom 50 in terms of popularity ("TIOBE Index | TIOBE - The Software Quality Company", 2018).

In relation to this it is interesting to note that one thing that both Swift and F# have in common is that they are both relatively newly created programming languages (Swift in 2014 and F# in 2005) ("Swift Has Reached 1.0 - Swift Blog", 2014) ("F Sharp (programming language)", 2018). As these two languages are the only ones in popular use that support physical quantities, it would suggest that supporting this functionality is something that needs to be built from the ground up in the language, as adding it to an already existing language (such as C# or Python) is either not possible or is so complex that it is not worth it.

There has also been a few attempts to make support for units of measurement a standard in the Java language. Specifically through the projects “JSR108”, “JSR275”, and “JSR363” ("The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 108", 2001) ("The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 275", 2010) ("The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 363", 2016). A similar proposal is the “UOMo” project (Guindon, 2010).

It is also possible to make the argument that C++ has support for physical quantities through the Boost::Units library which is one of many libraries that Boost.org provides for the language ("Chapter 43. Boost.Units 1.1.0 - 1.66.0", 2018). These libraries are all well made, well documented, and viewed by many as a de facto part of the C++ language.

Finally, there are some examples of smaller languages that support physical quantities, such as Nemerle ("About - Nemerle programming language official site", 2013) and Frink ("How Frink Is Different", n.d.). However, since these languages are rarely used, the fact that they do support this is not as relevant as the previous examples.

2.3 Related work

When looking at related work, it can mainly be divided into two groups. The first group is research which is more closely connected to the main goal of this paper, to look at available libraries, summarizing and categorizing them. The second group is research in the more general area, related to specific libraries or other solutions as well as theoretical approaches, without looking at the area as a whole.

2.3.1 Research in the specific area

As was briefly mentioned in section 1.3, there is little to no research that has been performed in the specific area. The largest academic contribution sofar is a talk given by Trevor Bekolay from the University of Waterloo at the 2013 SciPy conference (https://conference.scipy.org/). In his 20 minute presentation, entitled “A comprehensive look at representing physical quantities in Python”, Bekolay discusses why he thinks this type of functionality is essential for any language which sees heavy use in scientific applications (such as Python). He compares and contrasts 12 existing Python libraries that handle this, going through their functionality, syntax, implementation, and performance. In the end of the talk he also presents a possible “unification of the existing packages that enables a majority of the use cases” (Bekolay, 2013).

The way that Bekolay structured his research and his talk resembles this thesis in its structure. The major difference is that the scope of the thesis is much larger than that of Bekolay (although he analyzed the libraries he had chosen deeper). Another difference is that he only looked at libraries written in Python, resulting in his findings being far more limited compared to those presented here.

There has been some attempts at trying to summarize libraries in this area by the practitioners, e.g developers that are working on their own libraries. However, most of these are also limited, often only looking at libraries in the same language and only containing a name and a web address. These lists are mostly found in the documentation of physical quantity libraries hosted on GitHub, although some other examples exist:

- https://kdavies4.github.io/natu/seealso.html - 14 Python libraries listed
- https://github.com/cjrh/misu#inspiration - 14 Python libraries listed
- http://pint.readthedocs.io/en/latest/faq.html - 13 Python libraries listed
- https://archive.codeplex.com/?p=unitconversionlib - 13 .NET libraries listed
- http://juanreyero.com/open/magnitude/ - 3 Python libraries listed
- https://github.com/erussell/ngunits - 3 Java libraries listed
- https://github.com/spellman/quantity - 3 Java libraries listed

There are also a few places where one can find collections that that go beyond simply listing libraries in one language:

- https://github.com/szaghi/FURY#libraries - 14 libraries in different languages listed
- https://github.com/JustinLove/unit_conversion_shootout - compares 11 libraries against each other based on time performance
- https://github.com/martinmoene/PhysUnits-CT-Cpp11#other-libraries - 10 C++ libraries and 3 Python libraries listed
- https://github.com/timjb/quantities/wiki/Links - 10 libraries in different languages listed
- https://github.com/Shopify/measured#alternatives - 3 Ruby libraries listed, but includes pros and cons for each

And finally there is an article (albeit non-academic) in which three Python libraries are described and analyzed with a little more depth compared to the lists presented above (Hillar, 2013).

Judging from the fact that what is presented here is all the relevant research I have been able to find it is apparent that there is a lack of academic analysis of a more systematic kind in this area. This is a significant factor as to why the results of this thesis are important as there is a distinct knowledge gap that should be filled.

2.3.2 Research in the general area

In contrast to the lack of work in the specific area, there is a lot more that has been undertaken in the more general research area. I have chosen to present this research divided into two subgroups. The first group is research related to a specific software solution (such as a library or a package), whilst the other group is research that is more generally applicable in a wider context. Often relating to dimensional analysis theory in relation to software or describing certain algorithms.

I have also chosen to not describe the articles presented here in detail, instead opting to simply write the name and author (or authors). This choice was made as these articles are not always relevant to the main purpose of this thesis (as they are not part of the *specific* research area). The intention is that this section will serve a dual purpose, both giving a overview of the general research area as well as serving as a collection of relevant articles for any interested reader.

One example of an article in the group of research that revolves around some specific software is one in which Jiang and Su describe a type system ("Osprey") for automatic checking of potential errors involving units of measurement (Jiang & Su, 2006).

Other examples of research that fits into this group include:

1. *Automated computation and consistency checking of physical dimensions and units in scientific programs* (2001) by Grant W. Petty.

2. *A Java Extension with Support for Dimensions* (1999) by André van Delft.

3. *A Reusable Ada Package for Scientific Dimensional Integrity* (1996) by George W. Macpherson.

4. *A Rewriting Logic Approach to Static Checking of Units of Measurement in C* (2012) by Hills, Chen, and Rosu.

5. *An Ada Package for Dimensional Analysis* (1988) by Paul N. Hilfinger.

6. *Applied Template Metaprogramming in SIUnits: The Library of Unit-Based Computation* (2001) by Walter E. Brown.

7. *Introduction to the SI Library of Unit-Based Computation* (1998) by Walter E. Brown.

8. *Dimensional Analysis in Computer Algebra* (2001) by Raya Khanin.

9. *Fully Static Dimensional Analysis with C++* (1994) by Zerkis D. Umrigar.

10. *Lightweight Detection of Physical Unit Inconsistencies without Program Annotations* (2017) by Ore, Detweiler, and Elbaum.

11. *Object Oriented Units of Measurement* (2004) by Allen et al.

12. *Programming Pascal with Physical Units* (1986) by Dreiheller, Moersohbacher, and Mohr.

13. *Rule-Based Analysis of Dimensional Safety* (2003) by Chen, Rosu, and Venkatesan.

14. *Validating the Unit Correctness of Spreadsheet Programs* (2004) by Antonui, Steckler, and Krishnamurthi.

One of the pioneering works in the second group of research is an article by Wand and O'Keefe from 1991 in which they describe algorithms for integrating automatic dimensional analysis into modern compilers (Wand and O'Keefe, 1991).

Another author that has done a considerable amount of work in the area and extended Wand's work is Andrew J. Kennedy from the University of Cambridge. Some examples of articles he has written are:

- *Dimension Types* (1994)
- *Programming Languages and Dimensions* (1996)
- *Relational Parametricity and Units and Measure* (1997)
- *Formalizing an Extensional Semantics for Units of Measure* (n.d)

Kennedy has also written several blog posts about units of measure in the F# language (mentioned in Section 2.2).

Other notable work in the area has been made by Professor Jean Goubalt:

- *BDDs and Automated Deduction* (1994)
- *Proving with BDDs and Control of Information* (1994)
- I*nference d'unités physiques en ML* (1994)

and by Dr. Narain Gehani:

- *Units of Measure as a Data Attribute* (1976)
- *Databases and Units of Measure* (1982)

- *Ada's Derived Types and Units of Measure* (1985)
- *Dimensional Analysis with C++* (1988)

Other general examples include:

1. *A model-driven approach to automatic conversion of physical units* (2007) by Copper and McKeever.

2. *Conversion of units of measurement* (1995) by G.S. Novak.

3. *Conversion of Units of Measurement* (1997) by G.S. Novak.

4. *Arithmetic with measurements on dynamically-typed object-oriented languages* (2005) by Wilkinson, Prieto, and Romeo.

5. *A Proposal for an Extended Form of Type Checking of Expressions* (1983) by R.T. House.

6. *Certifying measurement unit safety policy* (2003) by Rosu and Chen.

7. *Dimension inference under polymorphic recursion* (1995) by Mikael Rittri.

8. *Error Detection and Unit Conversion: Automated Unit Balancing in Modeling Interface Systems* (2009) by Chizeck, Butterworth, and Bassingthwaighte.

9. *How semantics can improve engineering processes: A case of units of measure and quantities* (2011) by Rijgersberg, Wigham, and Top.

10. *Incorporation of units into programming languages* (1978) by Karr and Loveman.

11. *Quantities, units and computing* (2013) by Marcus P. Foster.

12. *Software Support for Physical Quantities* (2002) by B.D. Hall.

3 Methodology

3.1 Creating a physical quantity library

Before starting the process of finding and analyzing physical quantity libraries, I spent time on creating a library of my own. The main reason for doing this was because I realized that I had to immerse myself in the field and create my own implementation to better be able to understand the code I would later analyze as well as the approaches that other developers had taken. I had to 'learn by doing' as without this first-hand experience I would not be able to do this analysis as accurately.

This library was developed in C# and consisted of roughly 2600 lines of code. It is fully functioning (albeit with limited unit support) and is based on a class hierarchy using the seven base units in the SI system. It also uses dimensional exponents for all units.

The intention was not for the library to compete with all the ones that already exist but rather work as a prototype, a way to learn, as well as a way to look further into some specific functionality that could be implemented, specifically the idea of unit ranges (discussed in section 1.2). How unit ranges were implemented in this library is exemplified in Section 5.2.

3.2 Library search and analysis

Besides creating a physical quantity library, the main part of the research was spent finding, summarizing, and categorizing similar libraries found on open-source hosting sites. Because there is almost no previous research in this area (discussed in Section 2.3.1), the way this was carried out could not be copied from other researchers but instead had to be designed for this specific thesis with this goal in mind.

With this said, from a more general perspective the methodological approach can be likened to that of an exploratory study, or exploratory research. This type of research method is used when studying an area that has not been properly defined yet and in which very little or no previous research has been made. The result of exploratory research is meant to form the basis on which more conclusive research can be done later ("Exploratory Research - Research-Methodology", 2018). This is often done through reading related articles and conducting interviews but in the case of this thesis the main data collection has instead been looking for code repositories and the analysis is largely based on this code instead of text (although the documentation for each project has been an important part). Each step of this process is detailed below.

3.2.1 General search

The first objective in the library search process was to construct a list of keywords that could be used when looking for projects. Considering what was easily found on Google and GitHub a list of 10-15 relevant projects were initially found. Based on how the authors described these projects, eight keywords were chosen to describe the general area. These keywords were either used in the title of the projects that were found, in tags for these or in the documentation. The chosen keywords were:

- “Units of Measure”
- “Units Measure Converter”
- “Units”
- “Unit Converter”
- “Quantities”
- “Conversion”
- “Unit Conversion”
- “Physical Units”

After this, seven sites that host open source projects were chosen. The intention being to use the eight keywords described above on these seven sites to find as many projects as possible. These sites were chosen based on relevance and popularity (Hong, 2018), ("Top Source Code Repository Hosts: 50 Repo Hosts for Team Collaboration, Open Source, and More", 2017) , ("Comparison of source code hosting facilities", 2018). The sites were:

- GitHub.com
- BitBucket.org
- GitLab.com
- SourceForge.net
- CodeProject.com
- LaunchPad.net
- Savannah.gnu.org

Even though GitHub is the most extensively used out of these and could potentially have sufficed on its own, the intent was for the search to be as comprehensive as possible and therefore the other six sites were also included. This minimizes the risk that any software that could be of interest is missed.

This combination of sites and keywords returned over 65000 total (non-unique) projects, 78 percent of these being hosted on GitHub. Most of these results came from just two of the keywords on just one of the sites (GitHub), namely “Units” and “Conversion”. “Units” returning over 30000 results and “Conversion” over 17000. This is because they are the most general keywords in the list and also because in relation to software projects, “units” and “conversion” will relate to several things that are not relevant to the results of this thesis (such as conversion of data types).

Because this number of results was unrealistic and unnecessary to go through, a choice was made to put a cap of a maximum of 200 results (~20 pages) per keyword per site. Although this number is largely chosen arbitrarily, 200 results was seen as a good compromise between thoroughness and feasibility. The relevance of any projects outside of the first 200 results is generally very low and thus the chance of missing something important is slim to none. This cap made the total number of results go from 65000 down to about 3700, which was the final number of results that were checked and from which the most comprehensive libraries were chosen.

All the sites and search terms are listed in the two tables below and the number in each cell represents the total number of search results for that combination. If a cell has a number in parenthesis this means that the cap of 200 was employed and the number in the parenthesis is the total number of search results that would otherwise be available. For example, 200 (17110) for the keyword “Conversion” on the site GitHub means that 200 results for this keyword on this site were looked at but 17110 is the total amount. In the case of the site “LaunchPad.net” (see Table 2), the search function only showed the first 75 results. To indicate where this has affected the amount of search results “75 (Max)” has been written and under that the total number of results that would otherwise be available is shown.

Site > **Search Term** v	GitHub	BitBucket	GitLab	SourceForge	CodeProject
Units of Measure	200 (210)	15	1	17	97
Units Measure Converter	24	2	0	54	0
Units	200 (31019)	200 (1673)	69	200 (738)	200 (4979)
Unit Converter	200 (1369)	55	13	48	16
Quantities	200 (1586)	88	6	87	200 (772)

Conversion	200 (17110)	200 (778)	200 (222)	200 (3816)	200 (3087)
Unit Conversion	200 (822)	25	12	56	26
Physical Units	125	15	0	30	19
Sum	1349 - (52085)	600 - (2651)	32 - (323)	692 - (4846)	758 - (8980)

Table 1. Library keyword search results (1)

Site > **Search Term** v	LaunchPad	Savannah.GNU
Units of Measure	12	0
Units Measure Converter	2	0
Units	75 (Max) (254)	4
Unit Converter	11	0
Quantities	29	1
Conversion	75 (Max) (200)	10
Unit Conversion	4	0
Physical Units	8	1
Sum	216 - (520)	16 - (16)

Table 2. Library keyword search results (2)

3.2.2 Analyzing project validity

After the initial search, all the relevant projects that were found were analyzed again. During this second scrutinizing, some of these projects that initially looked relevant were deemed to be invalid and were therefore excluded.

A project was deemed invalid if at closer inspection it was lacking in one or several areas. These were for example that the code was incomplete, the documentation was severely lacking, it was not written in English, the code did not work, or that the solution was too limited in scope. Also, if the source code could not be accessed for whatever reason the project was not included as this was a requirement. The exact reason for each project being excluded can be found in appendix A, under the column marked “Reason for exclusion”.

3.2.3 Categorizing libraries and tools

When the list of valid projects was complete, those that were defined as ‘tools’ (described in section 1.4) rather than libraries were filtered out. Although the tool projects are not the main focus of this study, they can still be relevant to look at (albeit less comprehensively) as the algorithms or code used in these can also be useful in the context of a library. However, as these projects are of less relevance, they were filtered out into their own group. To view what repositories are part of this group, see appendix B.

3.2.4 Categorizing top tier and second tier libraries

In this final step, the libraries that had been filtered out in the previous step were divided into two groups which were labeled ‘top tier’ and ‘second tier’. As the name implies, these two groups reflect the quality of the libraries placed in each group, the top tier group being the best examples of physical quantity libraries.

Because the goal in this part of the process was to categorize the libraries in terms of *overall* quality (without having to look through every line of code), I attempted to triangulate ‘quality’ based on several factors of the library projects, namely:

- **The library is actively being worked on or has recently been updated** - If the library is actively being worked on and updated it generally means that the developers are continuously listening to requests and fixing bugs, resulting in a better and more comprehensive library.

- **The library has a high number of commits** - A library having a high number of commits generally means that it has been worked on for some time, bugs have been fixed, more functionality added, etc. Although the number of commits does not always correlate to quality (as the extent of what a commit is can vary), it is usually is a good measure of overall quality. Some libraries have as few as 5-10 commits while others have hundreds or even thousands. For the purposes of this thesis, a 'high number' of commits is seen as anything above ~100.

- **The library has a high number of contributors** - Similar to the point above, a high number of contributors can also be a good measurement of quality as this means many different people have worked on the project and added things of their own. Even though this is not a perfect measure of quality either, a project with 50 contributors will generally be more well-developed than a project with 1 contributor.

- **The library has comprehensive documentation** - Because I had to look at an extensive list of libraries written in many different programming languages I often had to rely on the documentation that was provided by the developers themselves, rather than being able to understand everything based on the code alone. This meant that the quality and comprehensiveness of the documentation became an important factor. This is also important for anyone else who wants to use the library, as a lack documentation means that it is harder to use and understand.

- **The library supports many different units and unit systems** - The ability for the library to support many different units and unit systems is a fundamental part of what these libraries are used for, and therefore a library that supports more units can generally be said to be of higher quality.

- **The library has high ratings and is popular** - High ratings (or something similar) is usually a good indication of overall quality. Similarly, a library that many people like and is popular is generally of good quality.

- **The library has some unique part, is written in an underrepresented language or in some other way sticks out** - Finally, if the library lacks some of the other qualities listed here but instead has a unique way of solving a related problem, provides a unique solution, or is written in an underrepresented language it can also be included in the top tier group. Although novelty is not necessarily a measure of quality I still chose to include some of these libraries as they provide interesting ideas for other developers to look at.

If a library is missing one or several of the criteria that would otherwise qualify it as a 'top tier' library it is instead placed in the 'second tier' group. The second tier group of libraries can be found in appendix C.

3.3 Top tier library analysis

After deciding on which libraries that fit in the 'top tier' group, these were analyzed in three different ways to be able to give further information about the group as a whole. The three ways these were analyzed were:

- Looking at how many units each library supports
- Categorizing each library from a code approach perspective
- Describing one library from each well-represented language more in-depth

These three analysis methods were chosen based on the fact that they give important information about the libraries to the reader while not requiring extensive analysis of each individual library. Striking a balance between depth and breadth when choosing what to look at was important as the goal of this research is to provide a summary of the area, not an in-depth analysis of each library.

These three steps are described in more detail in the sections below.

3.3.1 Unit support analysis

First, each library was analyzed in relation to how many units that each supports. For example if a library supports the use of 'meters', 'kilometers', 'feet', and 'yards' when measuring length this would mean that the library supports four units. Of course most libraries support many more than this.

As was described in the previous section, this functionality is fundamental to the main way that these libraries are used, and therefore an important factor when looking at the overall usefulness of a library in this area. Although this is the case, just a few projects have this information in their documentation. Therefore, for the vast majority of the libraries in this group, the code had to be manually checked and each unit counted and recorded.

3.3.2 Unit approach categorization

After this, each library was categorized based on if the units in the library code were defined *using dimensional exponents* or *not using dimensional exponents* (see Section 2.1.1). This categorization was chosen based on the fact that this gives relevant information on how each library works or is coded. Although there are many other ways of categorizing these libraries based on how they work (discussed further in Section 5.2), this was chosen as a good middle ground of giving valuable information to the reader while still being easy to ascertain from looking at the source code. Because the way the units are defined in the system is fundamental to the primary functionality, this categorization gets to the core of how each library is built up. To further exemplify these two categories of systems, two different libraries have been chosen and are presented below.

The first type of system uses dimensional exponents as a way to define their units. One example being the C# library "QuantitySystem" ("ibluesun/QuantitySystem", 2018). In this implementation, each unit has a corresponding "QuantityDimension" which contains this bit of code:

```
#region Dimension Physical Properties
public MassDescriptor Mass { get; set; }
public LengthDescriptor Length { get; set; }
public TimeDescriptor Time { get; set; }
public ElectricCurrentDescriptor ElectricCurrent { get; set; }
public TemperatureDescriptor Temperature { get; set; }
public AmountOfSubstanceDescriptor AmountOfSubstance { get; set; }
public LuminousIntensityDescriptor LuminousIntensity { get; set; }
#endregion
```

In this part of the code, the value for each of the seven base dimensions is kept track of, meaning that each unit can be represented as a combination of these seven values. Another example of this type of system is "Working with Units and Amounts" by Rudi Breedenraedt (Breedenraedt, 2013). The article that accompanies this project also explains the approach in a comprehensive manner.

The other library approach is defined in opposition to the first, in these libraries the units are defined *without* using dimensional exponents. An example of a library that employs this approach is another C# library, this one called "Cubico"("irperez/Cubico", 2017). In Cubico, each unit contains a reference to a "UnitType" and the "UnitType" class does not contain any values corresponding to dimensional exponents. Instead, all units are defined in an accompanying XML-file called "UnitData.xml".

In this file the different types are defined, for example:

```
<UnitType ID="1" Name="Temperature" />
<UnitType ID="2" Name="Mass" Description="Weight" />
<UnitType ID="3" Name="Time" />
<UnitType ID="4" Name="Force" />
<UnitType ID="5" Name="Momentum" />
<UnitType ID="6" Name="Velocity" />
<UnitType ID="7" Name="Length" />
<UnitType ID="8" Name="Power" />
<UnitType ID="9" Name="Energy" />
<UnitType ID="10" Name="Plane Angle" />
<UnitType ID="11" Name="Volume" />
```

and all units in the system are given a “UnitTypeID” in the same XML-file, corresponding to one of these groups. For example:

```
<Unit ID="1" Name="Kelvin" UnitTypeID="1" />
<Unit ID="11" Name="Kilogram" UnitTypeID="2" />
<Unit ID="25" Name="Second" UnitTypeID="3" />
<Unit ID="33" Name="Tonnes force" UnitTypeID="4" />
<Unit ID="47" Name="Metre" UnitTypeID="7" />
<Unit ID="88" Name="Cubic foot" UnitTypeID="11" />
```

3.3.3 Specific library analysis

The final step in the analysis process involved choosing one library from each of the most well-represented languages in the top-tier group (C#, Python, Ruby, C++, JavaScript, Scala, and Java). All of these languages had four or more libraries that were written in that language (see Section 4). This was deemed as a good cut-off point as choosing one from each language (30 total) would defeat the purpose of this type of analysis which is to give a good overview of the functionality that these libraries provide.

These seven libraries were chosen based on the same criteria that were generally used for selecting libraries for the top tier group (described above) but requiring even higher quality as to be able to narrow this down to just one library per language. On top of this criteria, I strived to get an even split of systems that use dimensional exponents and those that do not. As these libraries already are part of the ‘top-tier’ group they can be viewed as a type of ‘best-of-the-best’ group spread over the most well-represented languages found in the top tier group.

For each of these seven libraries the overall characteristics are first described. Going beyond this, three things are described for each library:

- How the units that come pre-defined in the library are implemented
- An example of how the library can be used (taken from the relevant documentation)
- How you add custom units to the library

These three areas were chosen as they complement the more general information and gives the reader a good idea of how these libraries work.

4 Results

The results of this thesis are divided into three sections. The first section gives general information about the top tier group of libraries as well as an overview of the process that was used to identify this group. This part gives the reader a summary of the whole top tier group without going into specifics.

The second section (4.2) is the main part of the results, listing the libraries that are part of the top tier group, grouped up according to which programming language each library is written in. These sections are in order of the most represented language to the least. The last subsection (4.2.10) contains all the languages for which there are only three or fewer libraries written in. Most of the languages in this section only include one library each.

In the third and final section seven different libraries, one from each of the seven most popular languages in the top-tier group are presented more in-depth.

4.1 General information

The overall search results included 3700 projects. From these, 586 projects were found to be relevant. Out of the 586 relevant projects, 391 were valid and looked at further.

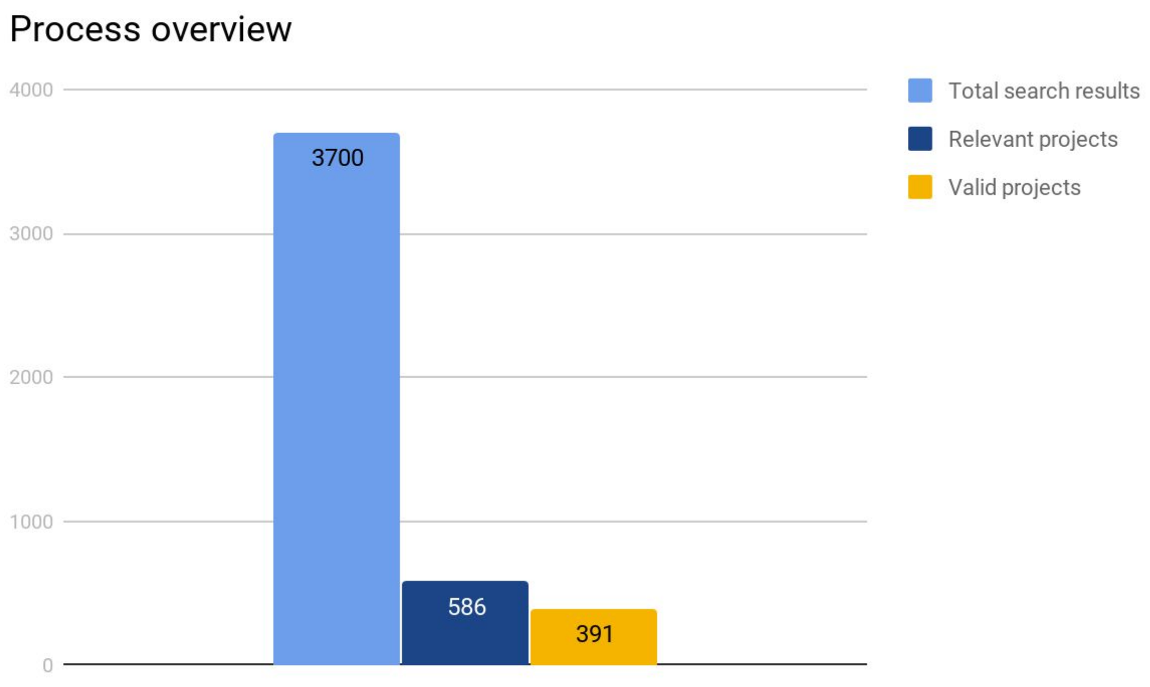


Fig 3. Library search process overview

From the group of 391 valid projects, 296 were libraries and 91 were tools.

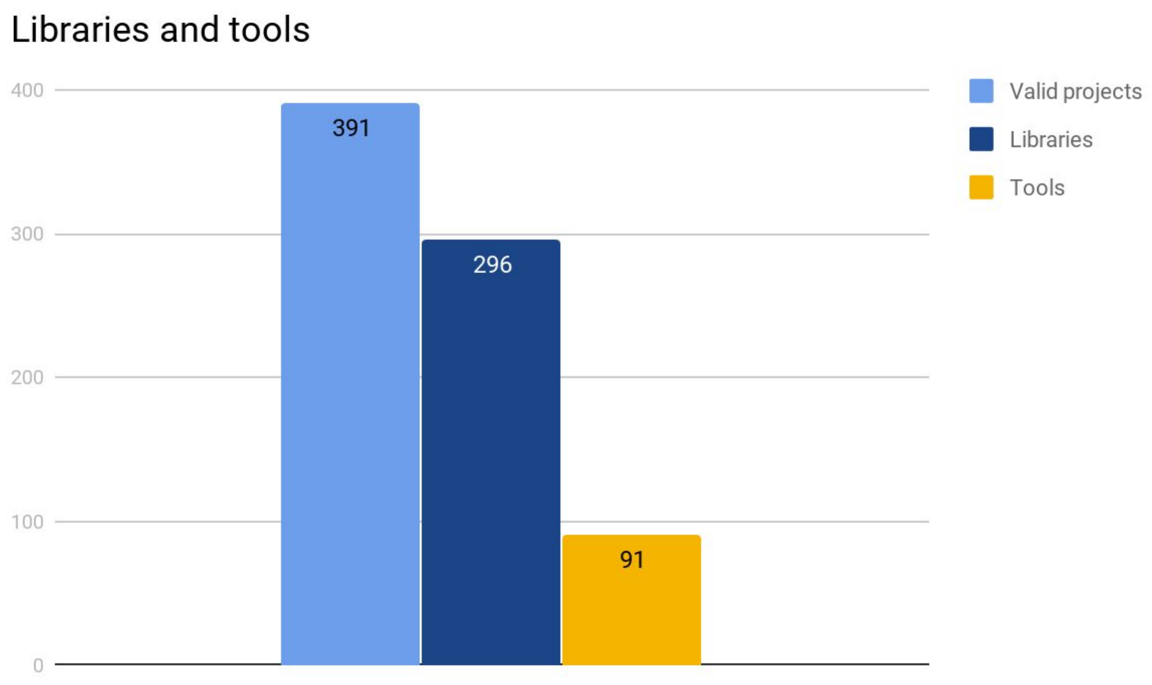


Fig 4. Valid projects split into libraries and tools

From the group of 296 libraries, 214 were deemed to be second tier and 82 to be top tier.

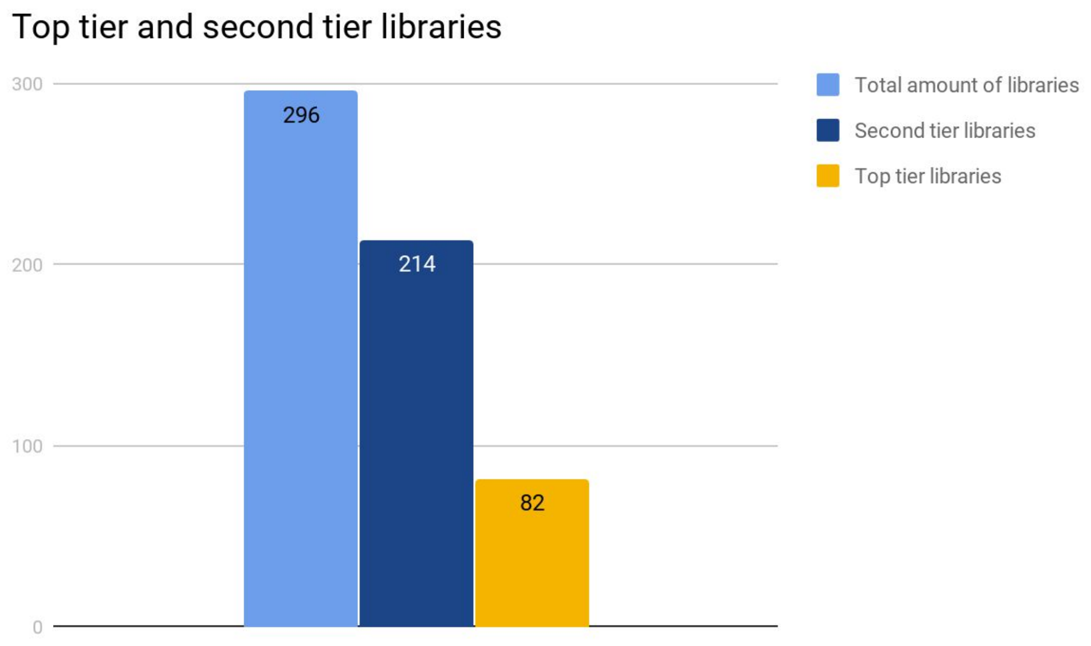


Fig 5. Libraries split into top tier and second tier

Of the libraries in the top tier group, approximately 80 percent were hosted on GitHub. Interesting to note is that this is almost identical to the amount of projects that were hosted on GitHub when looking at the group of total search results, which was 78 percent (see Section 3.2.1).

For the top tier group the total number of commits was about 36000, an average of about 440 commits per project. The average number of contributors is circa 8.3 per project. But these numbers are heavily influenced by the project 'Astropy' (https://github.com/astropy/astropy) which has almost 22000 commits and 240 contributors alone, making it an extreme outlier in the group. Removing the Astropy-data, the total number of commits drops to 14000 and the average commits per project to 175. The average number of contributors per project also goes down to ~5.5

75 percent of the projects were updated within the last two years (2017 and 2018), and many projects are being actively worked on and updated continuously. Looking at unit support, on average, each library supports about 200 unique units. Regarding programming language distribution in this group, there are 30 different languages in total and C# being the most popular, closely followed by Python and Ruby (see Figure 6).

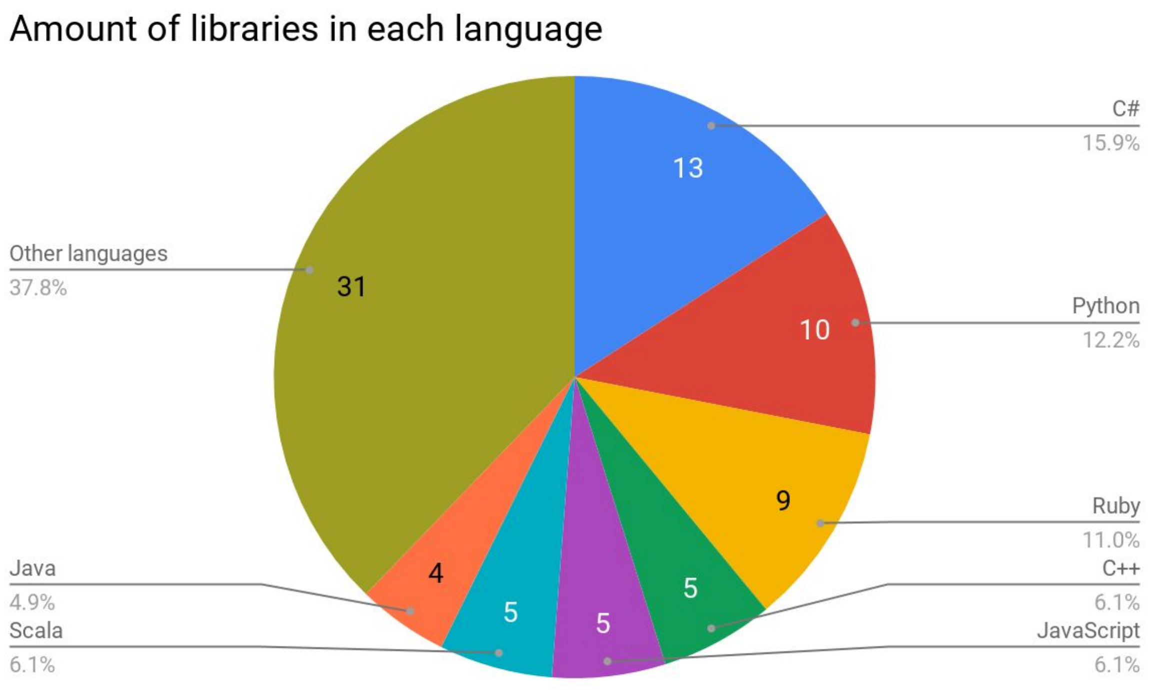


Fig 6. Distribution of programming languages used in the top tier library group

Finally, when it comes to the system type categorization there is an even split in this group, with 41 libraries using dimensional exponents in their definition of units and the other 41 not using exponents.

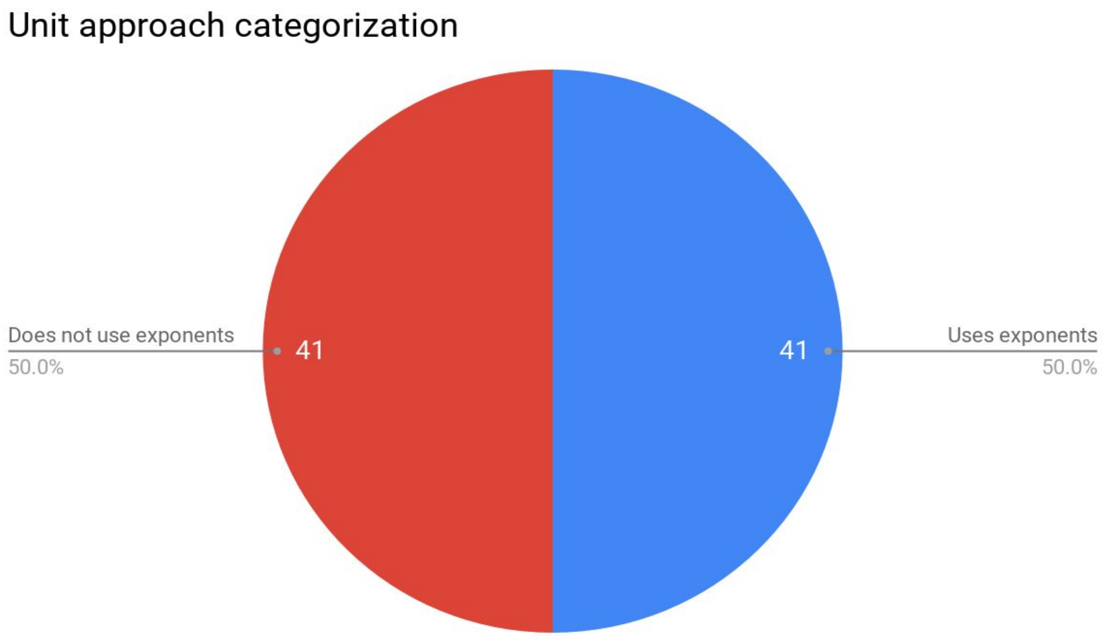


Fig 7. Unit approach distribution in the top tier group

4.2 Top tier libraries in each language

For each table below there are eight columns containing different information about each unique library:

- **Name:** The name of the library.
- **Latest update:** In what year the library was last updated.
- **Contributors:** The amount of unique contributors.
- **Commits:** The amount of commits (for a few libraries this is not clear).
- **Uses exponents:** If the library uses dimensional exponents to represent it's units (see Section 2.1.1 and 3.3.2)
- **Unit support:** How many units/prefixes/constants that the library supports by default.
- **Comments:** Any other comments about the library. These comments often contain information about what is especially interesting about the specific library.
- **Source/URL:** The link to where the code repository for the library can be found.

4.2.1 Libraries written in C#

Name	Latest update	Contri butors	Commits	Uses exponents	Unit support	Comments	Source / URL
UnitsNet	2018	60	1647	No	600+ units (including prefixes)	Large unit support. Continually updated and worked on by more than 50 people over many years. Comprehensive documentation. Localization of units.	https://github.com/angularsen/UnitsNet
Quantity System	2018	2	140	Yes	300+ units	Large amount of units. Contains a lot of currency units. Able to differentiate between Torque, Work, Angle, and Solid Angle. Actively updated.	https://github.com/ibluesun/QuantitySystem
Quantity Types	2017	12	340	No	300+ units	Large amount of units. Parsing to strings. Formatting to strings. Actively updated.	https://github.com/objorke/QuantityTypes
Working with Units and Amounts	2013	1	15	Yes	70+ units, 16 constants	Really good example of a dynamic solution using dimensional exponents (Yes). Comprehensive documentation.	https://www.codeproject.com/Articles/611731/Working-with-Units-and-Amounts
RedStar.Amounts	2018	1	68	No	~80 units, 16 constants	Can combine units to create new ones. Define your own units. Convert to and from JSON.	https://github.com/petermorlion/RedStar.Amounts/
Cubico	2017	1	90	No	~140 units	Over 2000 unit tests. Unit enum for easy browsing and finding of units. Easily extensible by editing an XML file. Focus on accuracy and precision.	https://github.com/irperez/Cubico
Using physical quantities and units of measure in C# programs	2015	1	2	No	~100 units	Comprehensive documentation.	https://www.codeproject.com/Articles/1066008/Using-physical-quantities-and-units-of-measurement
UnitParser	2018	1	23	Yes	~300 units	Large amount of units. Supports string inputs. Handles numeric values of any size.	https://www.codeproject.com/Articles/1211504/UnitParser

Fhir.Metrics	2018	2	43	Yes	~150 units, 24 prefixes	Supports uncertainty with error propagation. Does not implement temperatures	https://github.com/FirelyTeam/Fhir.Metrics
Gu.Units	2018	3	421	No	~50 units	Has a very large number of overloads for ToString() for units. Convert to and from JSON. Code generation when adding new units.	https://github.com/JohanLarsson/Gu.Units
CSUnits	2015	3	315	Yes	500+ units (including prefixes)	Large amount of units.	https://github.com/cureos/csunits
Convertinator	2015	1	34	No	~20 units	Conversions between units are defined as a bidirectional graph. This means that you don't have to explicitly define conversions for every possible combination of units - if a path between your two units exists, Convertinator can string together intermediate conversions to make it work.	https://github.com/hartez/Convertinator
CaliperSharp	2017	1	56	Yes	132 units, 16 constants, 23 prefixes	Supports the SI, US and British unit system. Able to create custom units. Supports unit name localization.	https://github.com/point85/CaliperSharp

Table 3. Top tier libraries written in C#

4.2.2 Libraries written in Python

Name	Latest update	Contributors	Commits	Uses exponents	Unit support	Comments	Source / URL
Pint	2018	80	958	Yes	300+ units, 20 constants, 27 prefixes	Large number of units. Very good documentation. Large number of contributors. Actively updated. Has support for different unit systems.	https://github.com/hgrecco/pint
Scimath	2018	13	313	Yes	300+ units	Large number of units.	https://github.com/enthought/scimath
Quantities (#1)	2017	18	455	Yes	300+ units, 28 prefixes	Large number of units.	https://github.com/python-quantities/python-quantities
Astropy	2018	241	21681	No	~150 units, 35 constants	Very good documentation. Supports a good amount of units. Actively updated. Part of a bigger project with a lot of people working on it (therefore the 241 contributors)	https://github.com/astropy/astropy
Efficient Scalars in Python	2008	1	11	Yes	~150 units	Good documentation, in a more academic style (10 pages). Able to shut off the scalar class in the library for added execution speed if needed.	http://russp.us/scalar-python.htm
ParamPy	2016	1	193	Yes	50+ units, 16 prefixes	Run with low overhead so it is suitable for use in simulations in which parameters will be evaluated millions of times. Iterate over a (potentially nested) range of values of different parameters, and then execute a provided function in the new parameter context. Lacking documentation but good solution.	https://github.com/matthewwardrop/python-parampy
Pyvalem	2017	2	48	Yes	70+ units	Good documentation. Easy to create new units.	https://github.com/xnx/pyqn
Quantiphy	2018	1	245	No	~40 units, 19 prefixes, 19 constants	Good documentation. Decent amount of units and constants. Actively updated.	https://github.com/KenKundert/quantiphy

Misu	2017	3	151	Yes	~450 units, 20 prefixes	"There are several units systems for Python, but the primary motivating use-case is that misu is written as a Cython module and is by far the fastest* for managing units available in Python." Focused on speedy execution. Large amount of units. Clear exception messages. Lots of redundancy when naming various units.	https://github.com/cjrh/misu
Aegon	2017	2	21	No	150+ units	Decent amount of unit support. Easy to add custom units.	https://github.com/lukaskollmer/aegon

Table 4. Top tier libraries written in Python

4.2.3 Libraries written in Ruby

Name	Latest update	Contributors	Commits	Uses exponents	Unit support	Comments	Source / URL
Ruby Units	2018	26	421	No	~150 units, 30 prefixes	Good amount of unit support. Ruby-units makes a distinction between a temperature (which technically is a property) and degrees of temperature (which temperatures are measured in). Emphasizes accuracy over speed. Actively supported.	https://github.com/olbrich/ruby-units
Unitwise	2017	4	257	No	300+ units	Very good amount of unit support. Built on UCUM. An expression grammar built with a PEG parser. Smart compatibility detection.	https://github.com/joshwlewis/unitwise
SY	2016	1	259	Yes	30+ units, ~10 constants	Focused on domain model correctness and ease of use.	https://github.com/boris-s/sy
Units-Ruby	2015	1	124	No	74 units, 5 prefixes	Good unit support.	https://github.com/bfoz/units-ruby
Uom	2011	1	28	Yes	63 units	Conversion between arbitrary	https://github.com/ca

						unit combinations. Custom unit definitions.	ruby/uom
Quantity	2011	3	85	Yes	~40 units, 25 prefixes,	Decent documentation. Can enable precision over speed by importing. "The abstractions go all the way down, and any conceivable conversion or munging functionality should be buildable on top of this."	https://github.com/bhuga/quantity
Alchemist	2017	12	150	No	500+ units, 88 prefixes	Very high amount of units supported. Contains really specific units, such as the length of the statue of liberty etc. Easy to create custom units.	https://github.com/halogenandtoast/alchemist
Unit_Soup	2017	1	26	No	0	Creates a graph of units from a set of rules	https://github.com/rutvij47/unit_soup
Phys-Units	2017	1	123	No	2300+ units, 85 prefixes	Crazy amount of unit support, best of any library BY FAR. Utilizes unit database from GNU Units.	https://github.com/masa16/phys-units

Table 5. Top tier libraries written in Ruby

4.2.4 Libraries written in C++

Name	Latest update	Contributors	Commits	Uses exponents	Unit support	Comments	Source / URL
BoostUnits	2018	?	?	Yes	~200 units, 20 prefixes, ~150 constants	Very good documentation, part of a big development (Boost.org). A LOT of constants. Good amount of unit support. Actively supported. De facto unit library in C++.	https://www.boost.org/doc/libs/1_66_0/doc/html/boost_units.html
Units (#1)	2018	10	364	Yes	~150 units, 22 prefixes	Good amount of units. Good documentation. Good compile time. No dependencies. Header only.	https://github.com/nholthaus/units
PhysUnits	2017	3	71	Yes	100+ units, 7 constants, 28 prefixes	No dependencies. Header only. Decent unit support.	https://github.com/martinmoene/PhysUnits-CT-Cpp11
Units (#2)	2017	1	48	Yes	~30 units, 330+ constants, 14 prefixes	Very high amount of constants supported. Specifically designed to be light weight and contain few lines of code compared to other libraries.	https://github.com/tonypilz/units
Units and measures for C++ 11	2017	1	20	Yes	~50 units	Very good documentation. Supports several unit systems.	https://www.codeproject.com/Articles/1088293/Units-and-measures-for-Cplusplus

Table 6. Top tier libraries written in C++

4.2.5 Libraries written in JavaScript

Name	Latest update	Contributors	Commits	Uses exponents	Unit support	Comments	Source / URL
Flowsnake	2018	1	167	No	650+ units	Very good unit support. Actively supported.	https://github.com/Eldelshell/flowsnake
JS-quantities	2018	21	329	No	~150 units, 30 prefixes	Good unit support. Decent documentation. Actively supported.	https://github.com/gentooboontoo/js-quantities
Mezur	2018	1	32	No	~100 units	"Building this is with intent to provide as much flexibility and extensibility as possible while keeping an easy-on-the-eyes usage". Actively supported.	https://github.com/guyisra/mezur
Unitz	2017	2	19	No	~70 units	Good documentation.	https://github.com/ClickerMonkey/unitz
Units by Stak Digital	2018	1	191	No	~50 units	Good documentation. Actively supported.	https://github.com/stak-digital/units

Table 7. Top tier libraries written in JavaScript

4.2.6 Libraries written in Scala

Name	Latest update	Contributors	Commits	Uses exponents	Unit support	Comments	Source / URL
Efficient Scalars in Scala	2015	1	8	Yes	~150 units	Good amount of unit support.	http://russp.us/scalar-scala.htm
Squants	2018	23	419	Yes	~300 units, 28 prefixes	Very good amount of unit support. Very good documentation. Actively supported.	https://github.com/typelevel/squants
Coulomb	2017	2	311	No	~70 units, 28 prefixes	Good documentation, contains tutorial.	https://github.com/erikerlandson/coulomb
SI -- A Scala Library of Units of Measurement	2017	1	57	No	~50 units, 20 prefixes	Decent documentation.	https://gitlab.com/h2b/SI
Scala-units	2014	1	144	Yes	~120 units	Good amount of unit support.	https://github.com/bwkimmel/scala-units

Table 8. Top tier libraries written in Scala

4.2.7 Libraries written in Java

Name	Latest update	Contributors	Commits	Uses exponents	Unit support	Comments	Source / URL
Unit API - JSR 385	2018	7	463	Yes	30+ units, 28 prefixes	Actively supported.	https://github.com/unitsofmeasurement/unit-api
Caliper	2017	1	133	No	~130 units, 23 prefixes, 16 constants	Good amount of unit support. Good documentation. Actively supported.	https://github.com/point85/caliper
SI-Units	2018	7	242	Yes	120+ units, 4 constants	Good amount of unit support.	https://github.com/unitsofmeasurement/si-units
JSR 363	2016	10	?	Yes	35 units	Very good documentation (45 pages!)	https://jcp.org/en/jsr/detail?id=363

Table 9. Top tier libraries written in Java

4.2.10 Libraries written in other languages

Name	Language	Latest update	Contributors	Commits	Uses exponents	Unit support	Comments	Source / URL
PHP Units of Measure	PHP	2017	14	204	No	~100 units	Good amount of unit support.	https://github.com/PhpUnitsOfMeasure/php-units-of-measure
UnitConverter	PHP	2017	1	19	No	130+ units	Good amount of unit support.	https://github.com/nfrazoo7/UnitConverter
Convertor	PHP	2018	4	70	No	70+ units	Easy to define your own units. Good documentation. Actively supported. No other dependencies.	https://github.com/olifolkerd/convertor
UOM-Plugin	Haskell	2017	5	225	No	0 units preloaded	Has an accompanying paper.	https://github.com/adamgundry/uom-plugin
Dimensional	Haskell	2018	3	610	Yes	~200 units, 21 prefixes	Good amount of unit support.	https://github.com/bjornbm/dimensional/blob/master/src/Numeric/Units/Dimensional/Quantities.hs
Quantities (#2)	Haskell	2015	1	79	No	200+ units, 28 prefixes, 3 constants	Good amount of unit support.	https://github.com/jdreaver/quantities
Physical Units for Matlab	MatLab	2018	1	80	No	800+ units	Very good unit support. Different unit systems.	https://github.com/sky-s/physical-units-for-matlab
Quantities (#3)	MatLab	2015	1	89	No	200+ units, 28 prefixes, 18 constants	Good amount of unit support. Supports uncertainty	https://github.com/mikofski/Quantities
UnitKit	Swift	2016	1	16	No	100+ units	Good amount of unit support. Only library in Swift.	https://github.com/otaviocc/UnitKit
MKUnits	Swift	2017	2	242	No	80 units	Decent unit support. Only	https://github.com/michalkonture

							library in Swift	k/MKUnits
Units (#3)	Multiple	2018	3	236	No	65 units, 70 constants	Supports multiple languages.	https://github.com/saroad2/units
PHYSICS	Multiple	?	1	?	Yes	~190 units, 25 prefixes	Good amount of unit support. Supports multiple languages.	http://sergey-l-gladkiy.narod.ru/index/physics/0-14
Units 2	Clojure	2017	1	49	Yes	~45 units, 28 prefixes	Dimensional analysis can be extended by the user (even at runtime!). Actively supported. Easy to define new units. Only library in Clojure.	https://github.com/mfey/units2
EiffelUnits	Eiffel	2002	1	?	Yes	~100 units, 21 prefixes	Good amount of unit support. Only library in Eiffel	http://se.inf.ethz.ch/old/projects/markus_keller/EiffelUnits.html
Unitful.jl	Julia	2018	20	440	Yes	~170 units, 50 prefixes, 21 constants	Good amount of unit support. Only library in Julia.	https://github.com/ajkeller34/Unitful.jl
Quantities (#4)	Idris	2017	3	102	Yes	~200 units, 24 prefixes	Good unit support. Only library in Idris.	https://github.com/timjb/quantities
Elixir Unit Converter	Elixir	2018	2	102	No	60+ units	Only library in Elixir. Good documentation.	https://github.com/carturoch/ex_uc
Units of Measure (#2)	Kotlin	2018	1	66	Yes	350+ units	Very good unit support. Only library in Kotlin.	https://github.com/kunalsheth/units-of-measure
Quantities (#5)	D	2018	2	228	Yes	~50 units, 20 prefixes	"The purpose of this small library is to perform automatic compile-time or run-time dimensional checking when	https://github.com/biozic/quantities

							dealing with quantities and units." Only library in D.	
Measures with Dimensions	Racket	2016	2	113	Yes	~150 units, 20 prefixes	Good amount of unit support. Only library in Racket.	https://github.com/AlexKnauth/measures-with-dimensions
Uncodium.Units	F#	2018	2	69	No	~350 units, 28 prefixes, ~50 constants	Interesting way to solve precision errors, can add 1 lightyear to 1 nanometer for example.	https://github.com/stefanmaierhofer/Uncodium.Units
Dimensioned	Rust	2017	3	179	Yes	~450 units & constants (hard to differentiate)	Very good amount of unit support. Different unit systems. Only library in Rust.	https://github.com/paholg/dimensioned
FURY	Fortran	2017	1	129	Yes	?	Only library in Fortran.	https://github.com/szaghi/FURY
PhysicalQuantities	Common Lisp	2017	1	159	No	~40 units, 28 prefixes	"User defined units including abbreviations and prefixes are supported. Error propagation and unit checking is performed for all defined operations."	https://github.com/mrossini-ethz/physical-quantities
Units of measurement for Ada	Ada	2018	1	?	Yes	100+ units, 20 prefixes	Good amount of unit support. Only library in Ada.	https://sourceforge.net/projects/unitsofmeasurementforada/?source=directory
Red-units	Red	2018	1	1	No	960 units, 1100+ constants, 100+ prefixes	Very high amount of unit support and constants. Only library in Red.	https://gitlab.com/maxvel/red-units
PPX_Measure	OCalm	2016	1	38	No	0 units preloaded. Generates a	Only library in OCalm	https://github.com/xvw/ppx_measure

						lot of code from simple commands instead (see documentation)		
Purescript-Quantities	PureScript	2018	1	106	No	~100 units, 23 prefixes, 14 constants	Good amount of unit support. Focuses on a representation at run time as opposed to other projects which use the type system to encode physical units at compile time.	https://github.com/sharkdp/purescript-quantities
Lua-Physical	Lua	2018	1	21	Yes	~150 units, 10 prefixes, 21 constants	Good amount of unit support. Only library in the Lua language. Supports uncertainty.	https://github.com/tjenni/lua-physical
Units (#4)	Tcl	2005	1	?	No	60 units, 21 prefixes	Only library in Tcl.	https://core.tcl.tk/tcllib/doc/trunk/embedded/www/tcllib/files/modules/units/units.html
Measurement Units for R	R	2018	6	443	Yes	Has 0 preloaded units itself, uses other Library "UDUNITS-2" for this	Only library in R.	https://github.com/r-quantities/units/

Table 10. Top tier libraries written in various ‘other’ languages

4.3 Specific library examples

4.3.1 'UnitsNet' (C#)

- **Repository link:** https://github.com/angularsen/UnitsNet
- **Programming language:** C#
- **Latest update:** May 2018
- **Amount of contributors:** 60
- **Amount of commits:** 1663
- **Comments:** Large unit support. Continually updated and worked on by more than 50 people over many years. Comprehensive documentation. Localization of units. Adding customized units.
- **Amount of unit support:** 600+
- **Uses exponents:** No
- **How are standard units defined:** Classes representing each unit are created from JSON objects using PowerShell scripts. There is an enum for each quantity that contains all available units of that quantity and for each quantity, a class is created that contains all the logic for conversions etc. As an example the class for acceleration is roughly 1200 LoC (lines of code).

- **Example usage:** In the example below the two classes "Mass" and "Length" are used to keep track of values that correspond to this quantity. You can also convert between different units using dot notation, such as converting a meter to a centimeter using the syntax *meter.Centimeters*.

```
// Convert to the unit of choice - when you need it
Mass weight = GetPersonWeight();
Console.WriteLine("You weigh {0:0.#} kg, weight.Kilograms);

// Avoid confusing conversions, such as between weight (force)
and mass
double weightNewtons = weight.Newtons // No such thing

Length meter = Length.FromMeters(1);
double cm = meter.Centimeters; // 100
double yards = meter.Yards; // 1.09361
double feet = meter.Feet; // 3.28084
double inches = meter.Inches; // 39.3701
```

- **Adding a custom unit:** Adding a custom unit in "UnitsNet" is difficult but possible. You either need to reuse the existing code generator scripts in a new project for your custom unit or add unit enum types at runtime and reuse the existing ToString() and Parse() methods. But the amount of units that are already included in the library by default means this is less of an issue.

4.3.2 'Pint' (Python)

- **Repository link:** https://github.com/hgrecco/pint
- **Programming language:** Python
- **Latest update:** May 2018
- **Amount of contributors:** 80
- **Amount of commits:** 974
- **Comments:** Large number of units. Very good documentation. Large number of contributors. Actively updated. Has support for different unit systems.
- **Amount of unit support:** 300+ units, 20 constants, 27 prefixes
- **Uses exponents:** Yes
- **How are standard units defined:** Defined in one large text file (default_en.txt).
- **Example usage:** In the example below a distance of magnitude 24.0 and the unit meter is created. Then a time quantity of 8 seconds is created. These two quantities can then be combined to create a third quantity, speed, by dividing the distance with the time.

```
>>> distance = 24.0 * ureg.meter
>>> print(distance)
24.0 meter
>>> time = 8.0 * ureg.second
>>> print(time)
8.0 second
>>> print(repr(time))
<Quantity(8.0, 'second')>

>>> speed = distance / time
>>> print(speed)
3.0 meter / second
```

- **Adding a custom unit:** Adding a custom unit is easy in 'Pint'. Either you can add a unit to a new text file and load it (this will persist). For example you can create a definition for a dog year in a file called "my_def.txt":

```
dog_year = 52 * day = dy
```

and then in Python, you can load it as:

```
>>> from pint import UnitRegistry
>>> # First we create the registry.
>>> ureg = UnitRegistry()
>>> # Then we append the new definitions
>>> ureg.load_definitions('/your/path/to/my_def.txt'
```

Or you can add it programatically (forgotten when the program ends):

```
>>> from pint import UnitRegistry
>>> # We first instantiate the registry.
>>> # If we do not provide any parameter, the default unit
definitions are used.
>>> ureg = UnitRegistry()
>>> Q_ = ureg.Quantity

# Here we add the unit
>>> ureg.define('dog_year = 52 * day = dy')

# We create a quantity based on that unit and we convert to
years.
>>> lassie_lifespan = Q_(10, 'year')
>>> print(lassie_lifespan.to('dog_years'))
70.23888438100961 dog_year
```

4.3.3 ‘Alchemist’ (Ruby)

- **Repository link:** https://github.com/halogenandtoast/alchemist
- **Programming language:** Ruby
- **Latest update:** August 2017
- **Amount of contributors:** 12
- **Amount of commits:** 150
- **Comments** Very high amount of units supported. Contains really specific units, such as the length of the statue of liberty etc. Easy to create custom units.
- **Amount of unit support**: 500+ units, 88 prefixes
- **Uses exponents:** No
- **How are standard units defined:** One .yml file (similar to XML) containing all definitions.

  Example usage: The simple example below showcases how one unit of length can be converted to another (miles into meters) as well as how units of the same quantity can be easily added to each other (miles being added to kilometers), without problematic conversions.

  ```
  8.miles.to.meters
  10.kilometers + 1.mile # 11.609344 kilometers
  ```

- **Adding a custom unit:** Adding a custom unit in ‘Alchemist’ is easy. Using the method “register” you can specify what quantity the unit is of, what it’s called as well as how it relates to an already existing unit (5 Ångströms in this case).

  ```
  Alchemist.register(:distance, [:beard_second, :beard_seconds],
  5.angstroms)
  ```

4.3.4 'BoostUnits' (C++)

- **Repository link:** https://www.boost.org/doc/libs/1_67_0/doc/html/boost_units.html
- **Programming language:** C++
- **Latest update:** April 2018
- **Amount of contributors:** ?
- **Amount of commits:** ?
- **Comments:** Very good documentation, part of a big development (Boost.org). A lot of constants. Good amount of unit support. Actively supported. De facto unit library in C++.
- **Amount of unit support:** ~200 units, 20 prefixes, ~150 constants
- **Uses exponents:** Yes
- **How are standard units defined:** They are defined as different headers that can be included if they are needed.
- **Example usage:** In the example below a quantity of type length (L) is created, this quantity is then combined with a quantity of mass as well as time to create a quantity of type energy (E). The length quantity is then used in different operations, showing examples of what can be done.

```
quantity<length> L = 2.0*meters;
quantity<energy> E = kilograms*pow<2>(L/seconds);
```

```
L                                 = 2 m
L+L                               = 4 m
L-L                               = 0 m
L*L                               = 4 m^2
L/L                               = 1 dimensionless
L*meter                           = 2 m^2
kilograms*(L/seconds)*(L/seconds) = 4 m^2 kg s^-2
kilograms*(L/seconds)^2           = 4 m^2 kg s^-2
L^3                               = 8 m^3
L^(3/2)                           = 2.82843 m^(3/2)
2vL                               = 1.41421 m^(1/2)
(3/2)vL                           = 1.5874 m^(2/3)
```

- **Adding a custom unit:** Adding a custom unit in 'BoostUnits' is difficult as the library is intended to "emphasize safety above convenience when performing operations with dimensioned quantities.". Therefore there are very high demands on any unit that is added, making it harder to do.

4.3.5 'JS-Quantities' (JavaScript)

- **Repository link:** https://github.com/gentooboontoo/js-quantities
- **Programming language:** JavaScript
- **Latest update:** March 2018
- **Amount of contributors:** 21
- **Amount of commits:** 329
- **Comments:** Good unit support. Decent documentation. Actively supported.
- **Amount of unit support:** ~150 units, 30 prefixes
- **Uses exponents:** No
- **How are standard units defined:** In one file (definitions.js).
- **Example usage:** In the example below several ways of creating a new quantity are shown. It is then shown how to convert m/h to ft/s by configuring a converter that corresponds to this. This converter can then be used for any value, or even an array of several values.

```
qty = new Qty('23 ft'); // constructor
qty = Qty('23 ft'); // factory

qty = new Qty(124, 'cm'); // => 1.24 meter
qty = Qty(124, 'cm'); // => 1.24 meter

var convert = Qty.swiftConverter('m/h', 'ft/s'); // Configures
converter

// Converting single value
var converted = convert(2500); // => 2.278..

// Converting large array of values
var convertedSerie = convert([2500, 5000, ...]); // =>
[2.278.., 4.556.., ...]
```

- **Adding a custom unit:** It is not clear from the documentation of this project how add custom units. But it should be possible to quite easily add a unit of your own to the file in which the standard units are defined (definitions.js).

4.3.6 'Squants' (Scala)

- **Repository link:** https://github.com/typelevel/squants
- **Programming language:** Scala
- **Latest update:** May 2018
- **Amount of contributors:** 23
- **Amount of commits:** 419
- **Comments:** Very good amount of unit support. Very good documentation. Actively supported.
- **Amount of unit support:** ~300 units, 28 prefixes
- **Uses exponents:** Yes
- **How are standard units defined:** Defines new classes in a number of files
- **Example usage:** The code below shows several examples of how physical quantities can be created and combined to create new quantities in 'Squants'. For example the variable "dodgeViper" which is of the type Acceleration and has the value 6.87 m/s^2, it being the acceleration a Dodge Viper car produces when going from zero to sixty miles per hour in 3.9 seconds.

```
scala> val dodgeViper: Acceleration = 60.miles / hour /
3.9.seconds
dodgeViper: squants.motion.Acceleration = 6.877552216615386
m/s²

val m: Length = Meters(10.22)
val km: Length = Kilometers(10.22)
val ft: Length = Feet(10.22)
val mi: Length = UsMiles(10.22)

val kg: Mass = Kilograms(10.22)
val speed: Velocity = UsMilesPerHour(55)

val length = 10 meters
val time = 4.2 hours
val speedLimit = 55.miles / hour
val milkPrice = 4.25.dollars / gallon
val energyUsed = 150.kilowatts * 3.5.hours
```

- **Adding a custom unit:** It is not clear from the documentation of this project how add custom units. And because the standard units are defined in separate classes, it is potentially rather time consuming to do this.

4.3.7 ‘Caliper’ (Java)

- **Repository link:** https://github.com/point85/caliper
- **Programming language:** Java
- **Latest update:** April 2018
- **Amount of contributors:** 1
- **Amount of commits:** 135
- **Comments:** Good amount of unit support. Good documentation. Actively supported.
- **Amount of unit support:** ~130 units, 23 prefixes, 16 constants
- **Uses exponents:** No
- **How are standard units defined:** In an enum, in one file (Unit.java).
- **Example usage:** In the example below is shown how quantities can be added, subtracted, and converted. As well as multiplied, divided, and inverted.

```
UnitOfMeasure m = sys.getUOM(Unit.METRE);
UnitOfMeasure cm = sys.getUOM(Prefix.CENTI, m);

Quantity q1 = new Quantity(2d, m);
Quantity q2 = new Quantity(2d, cm);

// add two quantities.  q3 is 2.02 metre
Quantity q3 = q1.add(q2);

// q4 is 202 cm
Quantity q4 = q3.convert(cm);

// subtract q1 from q3 to get 0.02 metre
q3 = q3.subtract(q1);

Quantity q1 = new Quantity(50d, cm);
Quantity q2 = new Quantity(50d, cm);

// q3 = 2500 cm^2
Quantity q3 = q1.multiply(q2);

// q4 = 50 cm
Quantity q4 = q3.divide(q1);
```

```
UnitOfMeasure mps = sys.getUOM(Unit.METRE_PER_SECOND);
Quantity q1 = new Quantity(10d, mps);

// q2 = 0.1 sec/m
Quantity q2 = q1.invert();

// A Tesla Model S battery has a capacity of 100 KwH.
// When fully charged, how many electrons are in the battery?
Quantity c = sys.getQuantity(Constant.LIGHT_VELOCITY);
Quantity me = sys.getQuantity(Constant.ELECTRON_MASS);
Quantity kwh = new Quantity(100, Prefix.KILO, Unit.WATT_HOUR);
Quantity electrons = kwh.divide(c).divide(c).divide(me);
```

- **Adding a custom unit:** Adding a custom unit in 'Caliper' is easy. You simply create a new "UnitOfMeasure"-instance, specifying the unit type, the name, and how this unit is derived. For example the unit "Gallons per hour":

```
// gallons per hour
UnitOfMeasure gph =
sys.createQuotientUOM(UnitType.VOLUMETRIC_FLOW, "gph",
"gal/hr", "gallons per hour", sys.getUOM(Unit.US_GALLON),
sys.getHour());
```

5 Discussion

5.1 Discussing the results

There are several things that can be interesting to discuss in relation to what has been presented in this thesis. First of all, what is the reason behind the fact that there are so many libraries and why so few of them seem used. When it comes to the first part of this question I think Bekolay put it quite well when he said that making a physical quantity library is easy but making a *good* one is hard (Bekolay, 2013). The core issue that these libraries aim to solve is something that is relatively simple to understand, common, as well as easy to create a simple solution for. This makes it well-suited as a hobby project or an assignment.

The problems arise when trying to make a more complete library, including more units, more operators, or more complex functionality overall. At this point it generally requires too much work for those doing this type of library as beginners or for fun. I think this is a major contributing factor as to why there are so many projects and so few truly comprehensive ones. In addition, there is quite a sharp decline in quality as soon as you look at projects outside of the top tier group, which supports this type of explanation.

Focusing more on the reasons for the lack of use I can mostly speculate (as this has not been the focus of this thesis) but one major reason could be that it can lead to significant performance issues while not not being relevant for most developers (Damevski, 2009). Another reason could be that the consequences of a mistake are underestimated and therefore it’s not seen as being worth it to go through the effort of using one of the available libraries.

A third reason that could have an effect is the general problem of accuracy that most libraries are unable to deal with. For example manifesting when trying to add quantities of vastly different magnitude to each other, such as a light year and an inch. These types of issues are hard to get away from as there are inherent inaccuracies that comes with using any kind of type to represent the value of a quantity, ("Floating-point arithmetic", 2018). This becomes especially problematic in relation to the type of scientific applications that these libraries are used for, as they often require exact precision. One example of a project that specifically aims to deal with this accuracy problem is “UnitParser”. According to the documentation for this library it is able to support numbers as big as $10^{2147483647}$ as well as over 27-digit precision (Garcia, n.d.). This is achieved through the implementation of a “mixed system” in the underlying unit class (Garcia, 2018).

In the interest of trying to go beyond the more speculative ideas presented above, I also conducted an interview with Oscar Grånäs - a researcher at the Department of Physics and Astronomy at Uppsala University ("Oscar Grånäs - Uppsala University, Sweden", 2018). His research focuses on materials theory, which includes the heavy use of software implementing physical quantities and units. During this interview, Grånäs mentioned that he and others in the departement have used the Python library "NumericalUnits", developed by Steven Byrnes (Byrnes, 2018). Interestingly, this is also one of the few libraries that Bekolay presents in his talk (Bekolay, 2013). The fact that Grånäs has conducted a lot of research in a highly relevant field as well as having experience with using physical quantity libraries makes him a well-suited interview subject.

When asked about the use of physical quantity libraries Grånäs said that is not as good as he would hope and that they should be used more, specifically in terms of trying to prevent costly mistakes. Looking at potential reasons for this lack of use he first brought up the impact on performance but mainly focused on a general lack of knowledge of the area.

The performance issues that these libraries can introduce was discussed previously in this section so this answer did not come as a surprise. To put this into perspective, Grånäs described how some calculations that he and his colleagues run require up to 100000 computation hours per month. With this in mind, it is easy to understand how even a relatively small time increase due to performance issues would have major implications for this type of work.

When discussing the research I have presented in this thesis, Grånäs seemed genuinely surprised about the amount of libraries that were found and that are being actively developed. From seeing this he concluded that the main reason for the lack of use of these libraries is that researchers such as him generally do not know about the fact that so many libraries exist and that getting more information about this is difficult. Even though Grånäs himself has used one of these libraries, he had little knowledge of what existed beyond this. At the same time he mentioned that the use of these types of solutions could become even more important in the future. His thoughts on the need to spread more information about these solutions gives further weight to the importance of the results presented in this thesis.

In Section 2.3, related work in the research area was presented and one thing that was made clear is that almost no one (save for one person) has done something similar to what has been presented in this thesis. The exact reason for this gap is not clear but potentially it could be because there has been a steady stream of new libraries being created and the academic field has not kept up with this development. Looking at the libraries available today, most of them are not more than a few years old, meaning that before this time it was a lot easier to keep track of those that existed but over the years it has now become a task that requires considerable time and effort.

Just like Bekolay did in his talk I would like to wrap up this discussion by looking at what I think could be the best way forward in terms of the development of these libraries, and specifically how the results of this thesis relates to this. Again, like Bekolay, I would suggest that one way forward is to try to focus the development efforts into one (or a few) projects.

One example of how this could be done is to create a solution that has multiple variations in different languages, something that has already been attempted, albeit on a smaller scale (see “Units (#3)” and “PHYSICS” in Section 4.2.10). This could potentially mean that there are several different teams that are responsible for the implementation into different languages but there is a common goal and communication between them. This type of effort could help with what seems to be happening at the moment, which is that a lot of effort is being put into developing existing libraries parallel to each other, without looking at what others are doing or have done.

Another suggestion is to combine this type of collaborative effort with the use of existing unit databases such as the one provided by UCUM (http://unitsofmeasure.org/trac) or GNU Units (https://www.gnu.org/software/units/). This way, development efforts can be focused on implementing the logic and implementation of a specific solution instead of defining all units and their relation to each other. To find examples of how this could be done one could look at “Phys-Units” (found in Section 4.2.3) which incorporates the GNU Units database into the library, making the developers able to support over 2300 units, far exceeding the unit capabilities of any other library in this group.

Finally, I think one the most important steps to take to develop this field further is to keep working on building up a general, shared, knowledge base. This way the existing physical quantity libraries can be developed in a more collaborative way. There are many articles related to this area (presented in Section 2.3.2) that can be used by library developers to better understand the problem or get inspiration in terms of specific algorithms. As there are no references to any of these articles (or any other) in the projects I’ve looked at, I can only surmise that this is not being done right now, creating a distinct gap between the academics and the developers even though they are both trying to solve the same issues.

5.2 Future directions

Looking at what was discussed in the previous section, one important future research direction is to look further into the lack of usage of these types of libraries, despite the abundance of them in many different languages. The discussion about this in the previous section gives some ideas as to why, but this is far from conclusive and thus there is a lot of room to investigate further. One idea is to interview more people who work heavily with programming scientific application about this phenomena (such as Oscar Grånäs) and through this type of research be able to see if the lack of knowledge is the main factor in relation to the lack of use or if there are other, underlying, issues that can be found.

Another suggestion is to do something similar to what has been presented in this thesis but analyze the libraries even deeper. As the goal for this study was breadth rather than depth there are many ways in which the analysis can be expanded. Several examples of what one might look at is presented by Bekolay (described in Section 2.3.1). One interesting categorization that I have not looked at is if the libraries mainly prevent errors during compile time or during runtime.

In relation to this, it would also be possible to expand this analysis further and look beyond just the libraries presented as the top tier group and also look at both tools and the second tier group. As I have not been able to do this kind of analysis there is a good chance that there are several libraries or tools that are better than they have appeared during my analysis. For example there could be projects that have been excluded because they lack documentation, but looking at their code deeper you might find that it is well-made, warranting a closer look. Because all of this data is also presented in appendix B and C, such an analysis would be possible without requiring time to search for libraries beyond what has been presented in Section 4.

Another interesting (albeit difficult) way to expand on the results presented here would be to attempt to make a 'best-of' solution, that takes the best part from several other libraries and integrates them into one. Because the top tier group is already filtered based on quality, this would mean that the main part of the foundation for such a solution could be found in this group. This type of approach could also be combined with the idea of a more collaborative effort that was discussed previously.

Finally one could look into successfully implementing support for ranges into existing libraries (this problem was described in Section 1.2). As was described in Section 3.1, this is something I specifically experimented with when making my own library. The way I chose to do it was to have a optional second and third parameter when creating a unit that would specify a minimum and maximum value.

When creating two new kilogram units it could look like this:

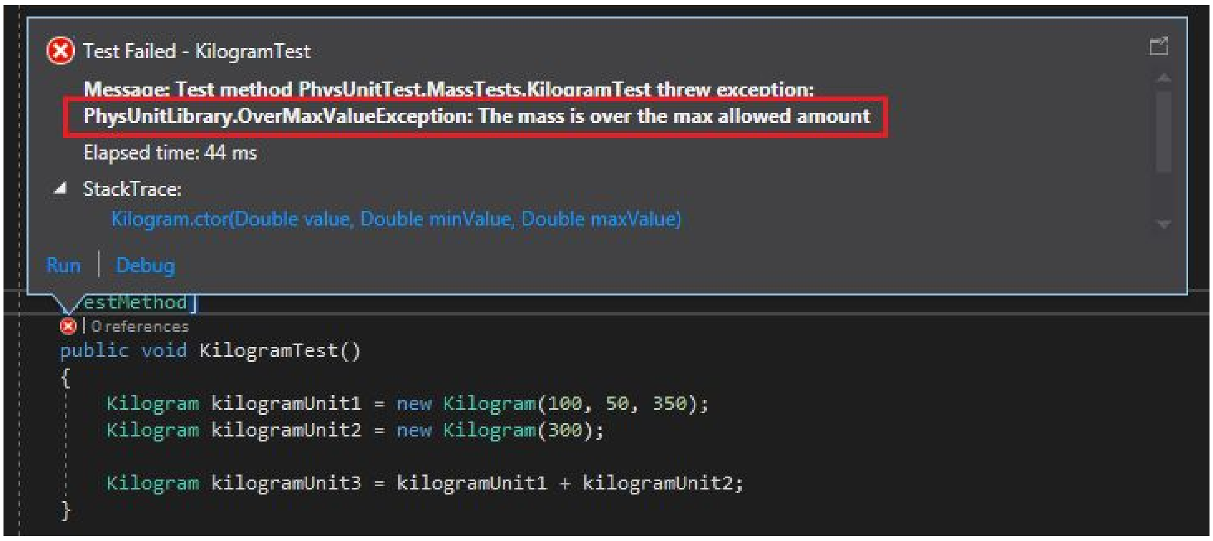
```
Kilogram kilogramUnit1 = new Kilogram(100, 50, 350);
Kilogram kilogramUnit2 = new Kilogram(300);
```

Here the value of "`kilogramUnit1`" is 100 kg and the value of "`kilogramUnit2`" is 300 kg. The difference is that for the first unit I've taken advantage of the optional parameters and set the minimum value of this unit to 50kg and the maximum to 350 kg.

So if we now try to add them together:

```
Kilogram kilogramUnit3 = kilogramUnit1 + kilogramUnit2;
```

The maximum value of 350 kg is exceeded and an exception of type "OverMaxValueException" is thrown, informing the user that the mass is over the max allowed amount (see Figure 8).



```
Test Failed - KilogramTest
Message: Test method PhysUnitTest.MassTests.KilogramTest threw exception:
PhysUnitLibrary.OverMaxValueException: The mass is over the max allowed amount
Elapsed time: 44 ms
StackTrace:
    Kilogram.ctor(Double value, Double minValue, Double maxValue)
```

```
[TestMethod]
0 references
public void KilogramTest()
{
    Kilogram kilogramUnit1 = new Kilogram(100, 50, 350);
    Kilogram kilogramUnit2 = new Kilogram(300);

    Kilogram kilogramUnit3 = kilogramUnit1 + kilogramUnit2;
}
```

Fig 8. Exceeding the max allowed amount

This functionality also works across conversions, meaning that if we did this:

```
Kilogram kilogramUnit1 = new Kilogram(100, 50, 350);
Kilogram kilogramUnit2 = new Kilogram(300);
Pound poundUnit = kilogramUnit1 + kilogramUnit2;
```

we would get the same error. The range is converted to the appropriate unit at the same time the object is. This means that the value of the two kilogram-units are added, converted to pounds, the range is converted and then then the program checks if the pound value exceeds the max range (in pounds).

It also works to add two units that both have ranges, the range for the new unit will then be the extent of the combined range of both units. For example, if we add these two units:

```
Kilogram kilogramUnit1 = new Kilogram(100, 50, 350);
Kilogram kilogramUnit2 = new Kilogram(100, 10, 300);

Kilogram kilogramUnit3 = kilogramUnit1 + KilogramUnit2;
```

The unit "`kilogramUnit3`" will now have the range 10-350 kg, meaning that the minimum value has to be over 10 kg and the maximum value under 350 kg. It has taken the maximum value from "`kilogramUnit1`" (as this is the largest of the two) and the minimum value from "`kilogramUnit2`" (as this is the smallest of the two).

Although the implementation described here is far from complete it still shows a way it can be done and interestingly there are very few libraries that I have found that have this type of functionality. "Squants" (presented in Section 4.2.6 and 4.3.6) is the only one that explicitly supports unit ranges. Besides this library, "UnitsNet" also implements maximum and minimum values for units (presented in Section 4.2.1 and 4.3.1). Because supporting ranges allows the libraries to solve another type of problem that is currently lacking support, it is something that can be worth to look at further and develop more.

6 Conclusions

In this thesis, a summary of 82 open-source, physical quantity libraries were presented. This group represents the most comprehensive and well-developed libraries, systematically chosen from approximately 3700 search results across seven sites. On such a scale, this type of study is the first of its kind.

There are thirty different programming languages represented in this group, meaning that there is considerable flexibility as to whom can find these results interesting. Out of these thirty languages, the three most popular are C#, Python, and Ruby.

In addition to presenting this 'top tier' group of libraries, they were also categorized in relation to how each library defines its units, with or without using dimensional exponents. This categorization was chosen based on the fact that it gives information about how the libraries fundamental part (the unit) is implemented, while still being easy to ascertain from viewing the source code. Interestingly, this was found to be an exact split into each group. 41 libraries utilized dimensional exponents and the other 41 did not.

This precise division also indicates that the categorization itself is well-chosen, as it captures the foundation upon which these libraries are built and thus all libraries in this area can be categorized in this way. This also means that it could be a good way to categorize all libraries in this field going forward, which is not something that is being done at the moment.

The results presented in this thesis are intended to help build a foundation upon which future developers can stand as this is currently missing and an overall more collaborative effort is needed. In addition to this, an absence of information about the area was indicated as being the primary reason for the general lack of use of these libraries, despite the abundance of them (as discussed in Section 5.1). If this is indeed the case, the results presented in this thesis could go a long way towards spreading this knowledge and making an important contribution to the field.

References

1. 15 U.S. Code § 205b - Declaration of policy. (2018). Retrieved from https://www.law.cornell.edu/uscode/text/15/205b [2018-06-04]

2. About - Nemerle programming language official site. (2013). Retrieved from http://nemerle.org/About [2018-06-04]

3. Bekolay, T. (2013). A comprehensive look at representing physical quantities in Python; SciPy 2013 Presentation. Retrieved from https://www.youtube.com/watch?v=N-edLdxiM40 [2018-06-04]

4. BIPM - dimensions. (2014). Retrieved from https://www.bipm.org/en/publications/si-brochure/section1-3.html [2018-06-04]

5. BIPM - quantities and units. (2014). Retrieved from https://www.bipm.org/en/publications/si-brochure/chapter1.html

6. Breedenraedt, R. (2013). Working with Units and Amounts - CodeProject. Retrieved from https://www.codeproject.com/Articles/611731/Working-with-Units-and-Amounts [2018-06-04]

7. Byrnes, S. (2018). numericalunits. Retrieved from https://pypi.org/project/numericalunits/ [2018-06-04]

8. Chapter 43. Boost.Units 1.1.0 - 1.66.0. (2018). Retrieved from https://www.boost.org/doc/libs/1_66_0/doc/html/boost_units.html [2018-06-04]

9. Clarke, R. (2016). Unit Systems in Electromagnetism. Retrieved from http://info.ee.surrey.ac.uk/Workshop/advice/coils/unit_systems/#sug [2018-06-04]

10. Cmelik, R., & Gehani, N. (1988). Dimensional Analysis with C++". IEEE Software.

11. Comparison of source code hosting facilities. (2018). Retrieved from https://en.wikipedia.org/wiki/Comparison_of_source_code_hosting_facilities

[2018-06-04]

12. Damevski, K. (2009). Expressing Measurement Units in Interfaces for Scientific Component Software (p. 1). Portland, OR, USA: ACM.

13. Dieterichs, H. (2012). Units of Measure Validator for C# - CodeProject. Retrieved from https://www.codeproject.com/Articles/413750/Units-of-Measure-Validator-for-Csharp [2018-06-04]

14. Essentials of the SI: Base & derived units. Retrieved from https://physics.nist.gov/cuu/Units/units.html [2018-06-04]

15. Exploratory Research - Research-Methodology. (2018). Retrieved from https://research-methodology.net/research-methodology/research-design/exploratory-research/ [2018-06-04]

16. Floating-point arithmetic. (2018). Retrieved from https://en.wikipedia.org/wiki/Floating-point_arithmetic#Accuracy_problems [2018-06-04]

17. F Sharp (programming language). (2018). Retrieved from https://en.wikipedia.org/wiki/F_Sharp_(programming_language) [2018-06-04]

18. Garcia, A. (2018). UnitParser - CodeProject. Retrieved from https://www.codeproject.com/Articles/1211504/UnitParser [2018-06-04]

19. Garcia, A. UnitParser overview (.NET/C#). Retrieved from https://customsolvers.com/en/pages/open_source/flexible_parser/unit_parser/overview/ [2018-06-04]

20. Guindon, C. (2010). UOMo | The Eclipse Foundation. Retrieved from https://www.eclipse.org/proposals/uomo/ [2018-06-04]

21. Hillar, G. (2013). Quantities and Units in Python. Retrieved from http://www.drdobbs.com/jvm/quantities-and-units-in-python/240161101 [2018-06-04]

22. Hong, N. (2018). Choosing a repository for your software project | Software Sustainability Institute. Retrieved from https://software.ac.uk/resources/guides/choosing-repository-your-software-pro

ject [2018-06-04]

23. How Frink Is Different. Retrieved from http://futureboy.us/frinkdocs/#HowFrinkIsDifferent [2018-06-04]

24. ibluesun/QuantitySystem. (2018). Retrieved from https://github.com/ibluesun/QuantitySystem [2018-06-04]

25. irperez/Cubico. (2017). Retrieved from https://github.com/irperez/Cubico [2018-06-04]

26. Jiang, L., & Su, Z. (2006). Osprey - A practical type system for validating dimensional unit correctness of C programs. ICSE 2006.

27. Joint Committee for Guides in Metrology (JCGM), *International Vocabulary of Metrology, Basic and General Concepts and Associated Terms* (VIM), III ed., Pavillon de Breteuil : JCGM 200:2012

28. Livsey, A. (2018). Scrambled in translation? Norway Olympics team orders 15,000 eggs by mistake. Retrieved from https://www.theguardian.com/world/2018/feb/08/norway-olympics-team-orders-15000-eggs-by-mistake-south-korea [2018-06-04]

29. Mars Climate Orbiter Fact Sheet. Retrieved from https://mars.jpl.nasa.gov/msp98/orbiter/fact.html [2018-06-04]

30. NASA. (1999). Mars Climate Orbiter Mishap Investigation Board Phase I Report.

31. Neumann, P. (1992). Illustrative risks to the public in the use of computer systems and related technology. ACM SIGSOFT Software Engineering Notes, 17(1), 23-32. doi: 10.1145/134292.134293

32. NSMeasurement - Foundation | Apple Developer Documentation. Retrieved from https://developer.apple.com/documentation/foundation/nsmeasurement [2018-06-04]

33. Oscar Grånäs - Uppsala University, Sweden. (2018). Retrieved from http://katalog.uu.se/profile/?id=N6-743 [2018-06-04]

34. Ranter, H. (2018). ASN Aircraft accident McDonnell Douglas MD-11F HL7373 Shanghai-Hongqiao Airport (SHA). Retrieved from

http://aviation-safety.net/database/record.php?id=19990415-0 [2018-06-04]

35. Sonin, A. (2001). The Physical Basis of DIMENSIONAL ANALYSIS. Department of Mechanical Engineering MIT Cambridge, MA 02139.

36. Swift Has Reached 1.0 - Swift Blog. (2014). Retrieved from https://developer.apple.com/swift/blog/?id=14 [2018-06-04]

37. The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 108. (2001). Retrieved from https://www.jcp.org/en/jsr/detail?id=108 [2018-06-04]

38. The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 275. (2010). Retrieved from https://jcp.org/en/jsr/detail?id=275 [2018-06-04]

39. The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 363. (2016). Retrieved from https://jcp.org/en/jsr/detail?id=363 [2018-06-04]

40. TIOBE Index | TIOBE - The Software Quality Company. (2018). Retrieved from https://www.tiobe.com/tiobe-index/ [2018-06-04]

41. Top Source Code Repository Hosts: 50 Repo Hosts for Team Collaboration, Open Source, and More. (2017). Retrieved from https://stackify.com/source-code-repository-hosts/ [2018-06-04]

42. Unit Mixups. (2018). Retrieved from http://www.us-metric.org/unit-mixups/ [2018-06-04]

43. Units of Measure in F#: Part One, Introducing Units. (2008). Retrieved from https://blogs.msdn.microsoft.com/andrewkennedy/2008/08/29/units-of-measure-in-f-part-one-introducing-units/ [2018-06-04]

44. Wand, M. and O'Keefe, P.M. (1991). Automatic Dimensional Inference. Computational Logic: in honor of J. Alan Robinson, MIT Press, pp. 479-486.

45. Witkin, R. (1983). JET'S FUEL RAN OUT AFTER METRIC CONVERSION ERRORS. Retrieved from https://www.nytimes.com/1983/07/30/us/jet-s-fuel-ran-out-after-metric-conversion-errors.html [2018-06-04]

Appendix A: Excluded projects

Name	Language	Latest update	Reason for exclusion	Source / URL
Udunitspy	Python	2016	The authors themselves recommend using CF_Units instead	https://github.com/blazetopher/udunitspy
Java Unit Conversion Library	Java	2003	Can't access source code	https://www.openhub.net/p/jucl
NUnitConverter	C#	2010	Not relevant	http://jamesnewkirk.typepad.com/posts/nunit_converter_v11.html
Unit Converter (#1)	C	2017	Test project	https://github.com/nithjino/UnitConverter
Unit	Ruby	2018	Only supports temperature	https://github.com/christinach/unit
FP Units	JavaScript	2017	Used for CSS units (pixels etc.)	https://github.com/anthonydugois/fp-units
Scientific Javascript	JavaScript	2017	Only a prototype	https://github.com/gkjohnson/scientific-javascript
Cerebro Converter	JavaScript	2018	Not relevant, only for "Cerebro"	https://github.com/KELiON/cerebro-converter
Quantity	Haskell	2018	No documentation	https://github.com/irreverent-pixel-feats/quantity
Cue	C++	2016	No documentation	https://github.com/OMGtechy/Cue
Units4j	Java	2017	No Documentation	https://github.com/echocat/units4j
Run-Convert	JavaScript	2018	Only for running metrics (pace, distance, speed, etc.)	https://github.com/cdbusby/run-convert
Trevor	JavaScript	2017	Incomplete solution	https://github.com/ozgrozer/trevor
TempConvert	Python	2018	Only for temperature	https://github.com/rootinchase/tempconvert
Temperature	Go	2018	Only for temperature	https://github.com/yanndr/temperature
SIUnits (#2)	C++	2018	No documentation	https://github.com/Jajauma/SIUnits

PhysicalUnits (#1)	F#	2017	No documentation	https://github.com/kainous/PhysicalUnits
Units (#10)	Python	2014	No documentation	https://github.com/DanielSank/units
Physical-Units	Scala	2016	No documentation	https://github.com/ppiotrow/physical-units
PhysicalQuantities (#3)	Python	2015	No documentation	https://github.com/ppfaff/PhysicalQuantities
UnitsOfMeasurement	C++	2017	Code in German, incomplete documentation	https://github.com/grafwolg/UnitsOfMeasurement
PhysicalQuantities (#4)	C#	2013	No documentation, incomplete solution	https://github.com/Traquina/PhysicalQuantities
Units.jl (#1)	Julia	2017	No documentation, says it's in "early development"	https://github.com/sclereid/Units.jl
PhysUnits (#2)	C++	2015	No documentation	https://github.com/rdlabspl/PhysUnits
PhysicalQuantities (#6)	Python	2018	No documentation, incomplete solution	https://github.com/docod3r/physicalQuantitis
Mensura	Python	2017	No documentation, incomplete solutionm, written in German	https://github.com/ypsilus/mensura
Comment-Units	Rust	2017	No documentation, incomplete	https://github.com/willi-kappler/comment_units
Elm-tunits	Elm	2017	No documentation, incomplete solution	https://github.com/gyulalaszlo/elm-tunits
Unit Conversion Wox Plugin	Python/PowerShell	2018	No documentation, incomplete solution	https://github.com/rpalo/wox-unit-converter
SI	C++	2013	Author recommends using boost.units instead	https://github.com/erdavila/SI
Units of measurement types in C++	C++	2014	Earlier version of "Units and measures for C++ 11". Same author.	https://www.codeproject.com/Articles/791511/Units-of-measurement-types-in-Cplusplus-Using-comp

A look at the Boost Units Library	C++	2015	An article	https://www.codeproject.com/Articles/988932/Boost-Units-Library
Measures and Units	C#	2013	No code	https://sourceforge.net/projects/measure/?source=directory
Converter Application	Python	2017	Only currency and temperature	https://sourceforge.net/projects/converter-application/?source=directory
Temperature Unit Converter	C	2015	Only temperature	https://sourceforge.net/projects/temp-unit-converter/?source=directory
XVertR - eXtensible units conVERTeR	JavaScript	2014	No documentation	https://sourceforge.net/projects/xvertr/?source=directory
Unit Converter (#3)	Java	2013	Only converts length, temperature, and fluidic volume	https://sourceforge.net/projects/unitsconverter/?source=directory
Imperial & Metric Converter	VB .NET	2013	Only supports some measurements	https://sourceforge.net/projects/exconverter/?source=directory
Aviation Tool	VB .NET	2016	Only for specific aviation measurements	https://sourceforge.net/projects/aviationtool/?source=directory
PHP Unit Converter	PHP	2015	Incomplete solution	https://sourceforge.net/projects/phpunitconvert/?source=directory
Spectroscopist's Energy Converter	C	2013	Only for energy	https://sourceforge.net/projects/specon/?source=directory
Winds Units Converter	C++	2016	Only for some units	https://sourceforge.net/projects/windunitsconver/?source=directory
Gas Flow Unit Conversion	Python	2016	Only for gas flow units	https://sourceforge.net/projects/gas-flow-unit-conversion/?source=directory
PUMA Repository	Pascal	2018	Not relevant	https://sourceforge.net/projects/puma-repository/?source=directory
Java Numbers with Unit	Java	2017	Solution in German	https://sourceforge.net/projects/jnumwu/?source=directory

QSAS	C++	2018	Contains too much other stuff on top of the unit conversion	https://sourceforge.net/projects/qsas/?source=directory
Rssfiz	Java	2017	No documentation	https://sourceforge.net/projects/rssfiz/?source=directory
EveryConverter	Java	2014	Limited solution	https://sourceforge.net/projects/everyconverter/?source=directory
UltimateCalculator	Java	2017	Too general (calculator)	https://sourceforge.net/projects/ultimatecalculator/?source=directory
Unit Conveter using PHP and HTML	PHP	2015	Only supports Area, Currency, Length, Volume, Weight, Temperature.	https://sourceforge.net/projects/stgmunitconvert/?source=directory
Imp Converter	C++	2013	Incomplete solution	https://sourceforge.net/projects/impconvert/?source=directory
Units Conversion Library	C	2013	No documentation	https://sourceforge.net/projects/units/?source=directory
DimensionalAnalysis	Python	2017	Only works for spreadsheets	https://sourceforge.net/projects/dimensionalanalysis/?source=directory
Scientific Library	C++	2014	More general solution	https://sourceforge.net/projects/scientificlib/?source=directory
Units (#12)	C++	2018	Same as Units (#6)	https://gitlab.com/eclufsc/eda/units
Easyunits-rs	Rust	2017	Limited solution	https://gitlab.com/imp/easyunits-rs
Fan measure	Fan	2012	No documentation, incomplete solution	https://bitbucket.org/qualidafial/fan-measure
Units (#13)	Go	2015	No documentation	https://bitbucket.org/ede/units
Units (#14)	Python	2013	No documentation	https://bitbucket.org/johanhake/units
Unit-converter (#2)	JavaScript	2017	Irrelevant	https://bitbucket.org/sergespaolonzi/unit-converter
All Unit Converter	Java	2015	Incomplete solution	https://bitbucket.org/Chawakorn_A/all-unit-converter

Unit Converter Application	Java	2014	No documentation	https://bitbucket.org/cpresley/unit-converter-application
Meshy-quantities	Python	2015	Only supports metres, inches, and yards	https://bitbucket.org/meshy/meshy-quantities/src
UnitConverterLibrary	C#	2015	No documentation	https://bitbucket.org/MADcod3r/unitconverterpublic
Measurement_Converter	Java	2017	Limited solution	https://github.com/KinYee/Measurement_Converter
UnitConverter (#2)	C++	2017	No documentation	https://github.com/Shaddox/UnitConverter
Unit-converter (#3)	Java	2017	Limited solution, no documentation	https://github.com/ThanasiG/unit-converter
CSharp-UnitsOfMeasure	C#	2016	Limited solution, no documentation	https://github.com/hediet/csharp-UnitsOfMeasure
UnitsOfMeasure (#1)	C++	2016	Limited solution, no documentation	https://github.com/ing200086/UnitsOfMeasure
UnitsOfMeasure (#3)	C++	2016	Limited solution, no documentation	https://github.com/printfn/UnitsOfMeasure
Measure	Perl	2014	Limited solution	https://github.com/bluefeet/Measure
Units-of-Measure (#2)	Java	2015	School assignment, not relevant	https://github.com/timroejr/2.05-Units-of-Measurement
UnitsDotNet	F#	2013	Only has length	https://github.com/benhamner/UnitsDotNet
Measurement with units	C++	2011	Limited solution	https://github.com/AndrewWasHere/measurements_with_units
Units-parser	Haskell	2017	Limited solution	https://github.com/adamgundry/units-parser
Measure	Swift	2017	Limited solution	https://github.com/sdrpa/measure
Measure.js	JavaScript	2018	No documentation	https://github.com/alnesjo/measure.js
Units (#20)	Elm	2017	Limited solution	https://github.com/irpagnossin/units
Unit (#4)	PHP	2015	Limited solution	https://github.com/komparu/unit
Open.Measuring	TypeScript	2014	No documentation	https://github.com/electricessence/Open.Measuring

Converter_Classes	Ruby	2014	Not relevant	https://github.com/gschool-g5/converter_classes
Kuants	Kotlin	2014	Only compatible with mass	https://github.com/evacchi/kuants
Units (#22)	JavaScript	2013	Only conversion between length and angle	https://github.com/heygrady/Units
SIUnits.jl	Julia	2017	Recommends Unitful.jl	https://github.com/Keno/SIUnits.jl
Units (#23)	Go	2015	Limited solution	https://github.com/alecthomas/units
UnitConverter (#6)	Java	2013	No documentation	https://github.com/sommukhopadhyay/UnitConverter
Unit-Converter (#2)	JavaScript	2013	Limited solution, no documentation	https://github.com/war1025/Unit-Converter
UnitConverter (#7)	Java	2015	No documentation	https://github.com/MarcKuniansky/UnitConverter
Converter (#6)	C++	2015	Limited solution	https://github.com/stevenharradine/Converter
UnitConverter (#7)	Java	2017	Limited solution	https://github.com/Ideally/UnitConverter
Unit-Converter (#4)	C++	2013	Only works for energy	https://github.com/BoboTiG/unit-converter
tConverter	PHP	2014	Only works for length and weight	https://github.com/tankotun/tConverter
Units-Converter(#5)	C#	2017	Unfinished	https://github.com/kolesnikova745/units-converter
UnitConverter (#8)	Java	2014	Only works for mass, speed and temperature	https://github.com/fnko/UnitConverter
Converter (#7)	Java	2016	Written in French	https://github.com/khaldi-yass/Converter
Unit (#5)	HTML	2015	Limited solution	https://github.com/Bingde/unit
UnitConverter (#9)	Python	2015	Only works for kilos to pounds	https://github.com/Demoleas715/UnitConverter
Simple-Unit-Converter	JavaScript	2016	Limited solution	https://github.com/smtaydemir/simple-unit-converter
UnitConverter (#11)	Java	2015	No documentation	https://github.com/suniltha lore/UnitConverter

Converter (#8)	Objective-C	2016	Limited solution, no documentation	https://github.com/milkyNik/Converter
Converter (#9)	Java	2017	Limited solution, no documentation	https://github.com/hirenjoo09/converter
Converter (#10)	Swift	2014	Limited solution, no documentation	https://github.com/dhunten/Converter
Converter (#12)	HTML	2017	Limited solution, no documentation	https://github.com/annaavanesyan/converter
Converter (#13)	JavaScript	2018	Limited solution, no documentation	https://github.com/Urrby/Converter
Converter (#14)	PureScript	2017	Limited solution, no documentation	https://github.com/moniyax/converter
Converter (#15)	Java	2014	Limited solution	https://github.com/NaOHman/Converter
Converter (#16)	Java	2016	Classed as "simple", no documentation	https://github.com/sunil512/Converter
Unit-Converter.js	JavaScript	2016	Very limited solution, no documentation	https://github.com/angeshpokharel/unit-converter.js
Converter (#17)	Java	2016	No documentation	https://github.com/jessepasos/Converter
Converter (#18)	JavaScript	2016	Only height and weight	https://github.com/zlargon/converter
Converter (#19)	C#	2017	No documentation, limited solution	https://github.com/sun-haha/Converter
Converter (#20)	JavaScript	2010	Experiment project	https://github.com/ziggy42/Converter
Converter (#21)	Java	2015	No documentation, only limited units	https://github.com/SurajPrasadd/Converter
Converter (#22)	Python	2017	Only units of distance	https://github.com/hpjardon/converter
Conv3rt3r	C#	2012	No documentation	https://github.com/torifat/Conv3rt3r
Converter (#23)	Java	2017	Only length	https://github.com/napnie/converter
Converter (#24)	Clojure	2012	Limited solution, no documentation	https://github.com/andreasfrom/converter
Converter (#26)	Python	2014	Irrelevant, no documentation	https://github.com/XTremeRox/converter

UniCon	Java	2018	No documentation	https://github.com/davoraleksic/UniCon
MilesMeter	Swift	2017	No documentation, only for metres and miles	https://github.com/mirokolodii/milesmeter
Unit-Converter	JavaScript	2018	Limited solution, irrelevant	https://github.com/jgalla/unit-converter
Unisum	Vue	2017	No documentation	https://github.com/askhat/unisum
Unit_Converter	Ruby	2016	No documentation	https://github.com/samuels410/unit_converter
Units (#26)	PHP	2017	Limited solution, no documentation	https://github.com/anstag/units
Convertoroid	Java	2018	No documentation	https://github.com/abhii2028/Convertoroid
Converter (#28)	Java	2017	Limited solution, no documentation	https://github.com/urielvan/converter
Converter (#29)	HTML	2016	Limited solution	https://github.com/AmitBuchnik/Converter
UnitConverter (#12)	Objective-C	2014	Incomplete solution, no documentation	https://github.com/basicsbeauty/UnitConverter
Groovy-Unitconverter	Groovy	2009	Incomplete solution	https://github.com/rhyolight/groovy-unitconverter
KingConverter	JavaScript	2014	Not relevant, Chrome extension	https://github.com/ryanr1230/KingConverter
UnitConverter (#13)	C#	2017	Limited solution, no documentation	https://github.com/github-ganesh/UnitConverter
Converter (#30)	Java	2015	Only for limited units	https://github.com/atsang36/Unit_Converter
Converter (#31)	C#	2017	No documentation	https://github.com/pfthroaway/Converter
TemperatureConverter	Python	2015	Only for temperature	https://github.com/bobo333/TemperatureConverter
BGConverter	C++	2017	No documentation	https://github.com/bugeaggeorge/BGConverter
Grobid-Quantities	JavaScript	2018	Irrelevant	https://github.com/kermitt2/grobid-quantities
Quantities-Comparison	Python	2015	Not relevant	https://github.com/tbekolay/quantities-comparison
Nim-Units	Nim	2017	Incomplete solution	https://github.com/def-/nim-units

Physical-Quantities	Haskell	2017	No documentation	https://github.com/mtesseract/physical-quantities
QuantitiesLib	C#	2017	No documentation, incomplete solution	https://github.com/isabellaalstrom/QuantitiesLib
Quantities	Haskell	2016	Incomplete solution	https://github.com/mtesseract/quantities
Quantities	JavaScript	2015	No documentation, incomplete solution	https://github.com/jdrew1303/quantities
RealizedQuantities	Python	2017	Incomplete solution	https://github.com/BayerSe/RealizedQuantities
JS-Quantities	JavaScript	2018	Only for parsing strings	https://github.com/mugendi/js-quantities
Scala-Quantities	Scala	2016	Code not working according to author	https://github.com/Lastik/scala-quantities
Units (#27)	C#	2018	No documentation	https://github.com/cardassianscot/Units
Quant	Python	2018	No documentation	https://github.com/colecocomo/quant
PhysicalQuantities	C#	2016	No documentation	https://github.com/KorzunK/PhysicalQuantities
Quantities.jl	Julia	2013	No documentation	https://github.com/ElOceanografo/Quantities.jl
Currency	Java	2018	Only currency	https://github.com/billthefarmer/currency
UnitConverter (#14)	Swift	2017	Some kind of student example, not relevant	https://github.com/SwiftEducation/UnitConverter
Ethereumjs-units	JavaScript	2016	Used to convert Etherum, not physical units	https://github.com/ethereumjs/ethereumjs-units
Rails-Units	Ruby	2015	Same as https://github.com/olbrich/ruby-units	https://github.com/scpike/rails-units
Hazpy.Unit-conversion	Python	2014	Focused on oil spills	https://github.com/NOAA-ORR-ERD/hazpy.unit_conversion
Convert	Python	2017	No documentation	https://github.com/sugarlabs/convert
Ktunits	Kotlin	2016	Only supports time and bytes	https://github.com/sargunv/ktunits

Silphid.Unity	C#	2018	Not relevant	https://github.com/silphid/silphid.unity
Physikal (#2)	Kotlin	2018	Forked from Physikal (#1)	https://github.com/Tenkiv/Physikal
NUCOS	JavaScript	2018	Specifically for oil spills	https://github.com/NOAA-ORR-ERD/NUCOS
Converge	Swift	2017	No documentation	https://github.com/daneden/Converge
Unit-Conversion	PHP	2017	Incomplete solution	https://github.com/h4kuna/unit-conversion
Ocalm-units	OCalm	2016	No documentation	https://github.com/pelzlpj/ocaml-units
Unit	Go	2017	Only conversion for data size (gigabite etc.)	https://github.com/kormoc/unit
Elm-Units	Elm	2016	Only has support for time and angle	https://github.com/anfelor/elm-units
Memory_Units	Rust	2018	Only units of memory	https://github.com/pepyakin/memory_units
Conversion (#3)	Java	2017	No documentation and incomplete solution	https://github.com/VaishnavRajesh/conversion
Conversion (#4)	Java	2014	No documentation	https://github.com/SoftronixGlobal/Conversion
Conversion (#5)	Java	2014	No documentation	https://github.com/2aredford/Conversion
Conversion (#7)	Java	2016	Only volume	https://github.com/ali-hamad/Conversion
Conversion (#8)	Java	2014	No documentation, a "test" project	https://github.com/ironwire/Conversion
Unitmaster	C++	2014	No documentation and incomplete solution	https://github.com/oswjk/unitmaster
ConversionsApp	Objective-C	2014	No documentation	https://github.com/JoeCortopassi/ConversionsApp
UnitConversion (#5)	Java	2015	Lacking	https://github.com/yadavparmatma/UnitConversion
UnitConversion (#7)	C#	2016	Incomplete solution	https://github.com/vsooraj/UnitsConversion

UnitComboLib	C#	2017	The project revolves around a combobox, not relevant	https://github.com/Dirkster99/UnitComboLib
ConversionApp2	Java	2015	No documentation and incomplete solution	https://github.com/dpinero/ConversionApp2
The Quantity Library	C++	2001	No access to sourcecode	http://home.xnet.com/~msk/quantity/quantity.html

Appendix B: Tools

Name	**Language**	**Latest update**	**Comments**	**Source / URL**
Phriky-Units	Python	2017	-	https://github.com/unl-nimbus-lab/phriky-units
ParamPool	Python	2016	-	https://github.com/hplgit/parampool
NumericalUnits	Python	2018	-	https://github.com/sbyrnes321/numericalunits
Rink	Rust	2018	-	https://github.com/tiffany352/rink-rs
Insect	PureScript	2018	-	https://github.com/sharkdp/insect
Units (#8)	JavaScript	2017	-	https://github.com/dohliam/units
Smart Units	Sidef	2016	-	https://github.com/trizen/smart-units
Chimp	Python	2017	-	https://github.com/mhetrerajat/chimp
XamConverter	C#	2017	-	https://github.com/brminnick/XamConverter
UnitConverter (#1)	Java	2018	-	https://github.com/nevack/UnitConverter
Convertee	JavaScript	2018	-	https://github.com/isquaredcreative/Convertee
Conversion Calculator	C++	2018	-	https://github.com/dtuivs/Converter
M2py	Python/Matlab	2015	-	https://github.com/caiorss/m2py
Scaler	Shell	2018	-	https://github.com/emugel/scaler
Praktool	Python	2013	-	https://github.com/zombofant/praktool
qMetrics	C++	2012	-	https://launchpad.net/qmetrics
SimpleUnitConverter	C	2009	-	https://launchpad.net/simpleunitconverter
Enhancing User Experience - Part 1: A Simple Unit Converter Application	C#	2004	-	https://www.codeproject.com/Articles/6667/Enhancing-User-Experience-Part-A-Simple-Unit-Con
Libre Unit Conveter	C#	2017	-	https://sourceforge.net/projects/libreunitconverter/?source=directory
ConvertAll	Python	2017	-	https://sourceforge.net/projects/convertall/?source=directory
JFX Konwerter	Java	2017	-	https://sourceforge.net/projects/jfx-konwerter/?source=directory
Unit Converter (#2)	Scratch	2016	-	https://sourceforge.net/projects/unitconverter12/?source=directory
MAIA Unit Converter	C++	2013	-	https://sourceforge.net/projects/maia-unit-conv/?source=directory

Unit Conveter (#3)	C#	2013	-	https://sourceforge.net/projects/unitconversion/?source=directory
Converter (#2)	Java	2012	-	https://sourceforge.net/projects/con-vert/?source=directory
LibreEngineering	Python	2013	-	https://sourceforge.net/projects/libreeng/
UNeedIT Converter	C#	2013	-	https://sourceforge.net/projects/uneedconverter/?source=directory
Unit Converter (#4)	Python	2014	-	https://sourceforge.net/projects/unit-converter/?source=directory
Portable Unit Converter	JavaScript	2013	-	https://sourceforge.net/projects/puc/?source=directory
Web Unit Converter	C#	2017	-	https://sourceforge.net/projects/web-unit-converter/?source=directory
ChemicalA	C++	2017	-	https://sourceforge.net/projects/chemicala/?source=directory
Computing with Units	Java	2014	-	https://sourceforge.net/projects/units-in-java/?source=directory
EngineeringCalculator	C++	2016	-	https://sourceforge.net/projects/alwaysontopcalc/?source=directory
Quick Unit Converter	Java	2016	-	https://sourceforge.net/projects/qunitconverter/?source=directory
Unit Converter (#5)	REBOL	2014	-	https://sourceforge.net/projects/tbunitconverter/?source=directory
ConverTo	C++	2015	-	https://sourceforge.net/projects/converto/?source=directory
UConverter	Java	2016	-	https://sourceforge.net/projects/uconverter/?source=directory
GodSend	PHP	2013	-	https://sourceforge.net/projects/godsend/?source=directory
JConvert	Java	2015	-	https://sourceforge.net/projects/jconvert/?source=directory
SIMConverter	C#	2013	-	https://sourceforge.net/projects/simconverter/?source=directory
Esos	C++	2013	-	https://sourceforge.net/projects/esos/?source=directory
Punits - Pluggable units	C	2013	-	https://sourceforge.net/projects/punits/?source=directory
BeConverter	C++	2016	Documentation @ https://sites.google.com/site/appbeconverter/home	https://bitbucket.org/Teknomancer/beconverter

Unit (#2)	Java	2017	-	https://bitbucket.org/hansolo/unit
Unit-converter (#1)	Python	2018	-	https://bitbucket.org/brayvasq/unit-converter
Physics	Python	2017	-	https://bitbucket.org/hppavilion1/physics
Frinj	Clojure	2015	-	https://github.com/martintrojer/frinj
Converter (#3)	Java	2017	-	https://github.com/samWson/converter
SimpleMeasurementConverter	Java	2013	-	https://github.com/michaelstoops/SimpleMeasurementConverter
Converter (#4)	C++	2017	-	https://github.com/dtalaba/convertor
Frins	Scala	2016	-	https://github.com/martintrojer/frins
MeasurementConverter	Java	2015	-	https://github.com/SBrooks75/MeasurementConverter
UnitConverter (#4)	Java	2017	-	https://github.com/MarioDudjak/UnitConverter
Maser	JavaScript/Python	2014	-	https://github.com/justinbangerter/maser
JC-Units	Python	2017	-	https://github.com/jasonox43/jc-units
UnitConverterUltimate	Java	2017	-	https://github.com/physphil/UnitConverterUltimate
Units (#25)	Java	2017	-	https://github.com/xxv/Units
Unit-Converter (#1)	JavaScript	2011	-	https://github.com/TheMarco/Unit-Converter
Alfred-Converter	Python	2017	-	https://github.com/WoLpH/alfred-converter
UnitConverter.htm	JavaScript	2018	-	https://github.com/iterami/UnitConverter.htm
UnitConverter (#7)	Java	2018	-	https://github.com/impateljay/UnitConverter
Unit-Converter (#3)	Java	2012	-	https://github.com/AmitKumarSah/Unit-Converter
PythonUnitConverter	Python	2017	-	https://github.com/Aaron1011/PythonUnitConverter
Converter (#5)	Java	2017	-	https://github.com/HanSolo/converter
Mather	Java	2017	-	https://github.com/icasdri/Mather
Vandelay	C++	2017	-	https://github.com/HaikuArchives/Vandelay
Unit-Converter (#3)	PHP	2015	-	https://github.com/b-sebastian/unit-converter
Unit-Converter (#4)	C++	2016	-	https://github.com/mikeleppane/Unit-Converter

Unit-Converter (#6)	JavaScript	2014	-	https://github.com/MaicolBen/unit-converter
Unit	PHP	2012	No documentation, but OK solution	https://github.com/pounard/Unit
Converter (#11)	C++	2017	-	https://github.com/KerryL/Converter
Angular-Converter	JavaScript	2014	-	https://github.com/cyrilf/angular-converter
PreciseUnitConverter	JavaScript	2012	-	https://github.com/bumxu/PreciseUnitConverter
Converter (#25)	TypeScript	2017	-	https://github.com/ialex90/Converter
DakaUnitConverter	Java	2017	-	https://github.com/darrylfonseka/DakaUnitConverter
Convertor	C++	2017	-	https://github.com/mihaelateo/Convertor
Converter (#27)	C++	2016	-	https://github.com/scruff3y/Converter
Unit-Converter	JavaScript	2018	-	https://github.com/GTLook/Unit-Converter
UnitMan	C#	2017	-	https://github.com/mbeloshapkin/UnitMan
PyUnitConverter	Python	2017	-	https://github.com/jyri78/PyUnitConverter
All_In_One_Converter	Python	2018	-	https://github.com/pradeepjairamani/All_in_one_converter
ConverterApp	C#	2017	-	https://github.com/halezmo/ConverterApp
Alfred-Convert	Python	2018	-	https://github.com/deanishe/alfred-convert
Xiny	Go	2018	-	https://github.com/bcicen/xiny
GenUnitApp	JavaScript	2018	-	https://github.com/halcwb/GenUnitApp
Unitconverter-Android	Java	2018	-	https://github.com/dbrant/unitconverter-android
Unit_Conversion	Python	2018	-	https://github.com/bastihaase/unit_conversion
UnitConversion (#6)	Java	2016	-	https://github.com/yflo07/UnitConversion
Unit-Conversion	JavaScript	2017	-	https://github.com/srimanthk/unit-conversion
GNU Units	C#	2017	Documentation @ https://www.gnu.org/software/units/manual/units.html	https://www.gnu.org/software/units/

Appendix C: Second Tier Libraries

Name	Language	Latest update	Contributors	Commits	Comments	Source / URL
Convert Units (#1)	Java	2018	24	187	Might be tier 1	https://github.com/ben-ng/convert-units
Unit (#1)	Go	2018	1	34	Lacking documentation	https://github.com/martinlindhe/unit
UnitConversionLib	C#	2014	1		Several better examples in C#	https://www.codeproject.com/Articles/787029/UnitConversionLib-Smart-Unit-Conversion-Library-in
Natu	Python	2017	3	67	In a "pre-release state"	https://github.com/kdavies4/natu
Buckingham	Python	2016	1	4	Limited solution, commits, contributors	https://github.com/mdipierro/buckingham
Magnitude (#1)	Python	2015	1	34	More documentation @ http://juanreyero.com/open/magnitude/	https://github.com/juanre/magnitude
Units (#2)	Python	2017	3	81	Documentation @ https://bitbucket.org/adonohue/units/src/b187fbf3c1d403df9326670e7a00e2522fc74215/units/__init__.py?at=default&fileviewer=file-view-default	https://bitbucket.org/adonohue/units
Unum (#1)	Python	2018	5	38	Lacking documentation	https://bitbucket.org/kiv/unum
CF_Units	Python	2018	13	118	Might be tier 1	https://github.com/SciTools/cf_units
DimPy	Python	2008			Older solution	http://www.inference.org.uk/db410/dimpy/docs/docs/docs.html
Units of Measure (#1)	C#	2017			Don't have easy access to source code, lacking documentation	https://archive.codeplex.com/?p=unitsofmeasure
Physical Measure	C#	2016	1	11	Several better examples in C#, limited commits	https://github.com/KiloBravoLima/PhysicalMeasure
NGenericDimensions	C#	2017	1	18	Several better examples in C#, limited commits	https://github.com/MafuJosh/NGenericDimensions

Quantities.net	C#	2014	1		No documentation	https://sourceforge.net/projects/quantitiesnet/
.NET Unit Conversion Library	C#	2014	2		No documentation	https://sourceforge.net/projects/unitcon/
Measurement Unit Conversion Library (#1)	C#	2010	1	3	Comprehensive documentation but lacking unit support (40 units)	https://www.codeproject.com/Articles/23087/Measurement-Unit-Conversion-Library
Unit of Measure Library for .NET	C#	2014	1	6	This library is part of the Units of Measure Validator for C#.	https://www.codeproject.com/Articles/404573/Units-of-Measure-Library-for-NET
Unit of Measure Validator for C#	C#	2012	1	2	Comprehensive documentation but lacking unit support (15 units only)	https://www.codeproject.com/Articles/413750/Units-of-Measure-Validator-for-Csharp
Units of Measurement Systems	Java	2017	5	381	Lacking documentation	https://github.com/unitsofmeasurement/uom-systems
Units (#3)	Java	2017	1	20	Not as comprehensive as other solutions	https://github.com/SamCarlberg/units
Units (#4)	C#	2017	1	16	Not as comprehensive as other solutions	https://github.com/engineers-tools/Units
PHP Unit Conversion	PHP	2017	2	22	Interesting "nearest" function	https://github.com/pimlie/php-unit-conversion
The Units of Measure Library	C++	2013	1		In "beta"	https://sourceforge.net/projects/tuoml/
Metric (#1)	Rust	2017	2	67	Not as comprehensive as other solutions	https://github.com/coder543/metric
ScalaU	Scala	2013	1	7	-	https://github.com/adrianfr/scalau
Units (#5)	Go	2016	1	11	Does lack support for some units but good enough for second pass anyway	https://github.com/smyrman/units
Superquants	Scala	2017	1	4	Pre-release	https://github.com/rudogma/scala-superquants
Metric (#2)	Nim	2018	1	4	Limited documentation, limited commits	https://github.com/mjendrusch/metric
Units D	D	2017	1	14	Unfinished	https://github.com/nordlow/units-d
Units and Measurements	Racket	2018	3	65	Unfinished solution (untested)	https://github.com/Metaxal/measures

SIUnits (#1)	Haskell	2018	1	117	Not as comprehensive as other solutions	https://github.com/joewkr/SIUnits
Physikal	Kotlin	2017	3	83	Lacks documentation	https://github.com/Tenkiv/Physikal
Engineering	C#	2017	1	35	Not as comprehensive as other solutions	https://github.com/zhofre/engineering
ProgParam	C++	2017	1	23	Not as comprehensive as other solutions	https://github.com/qPCR4vir/ProgParam
InkUnits	Swift	2017	1	10	Not as comprehensive as other solutions	https://github.com/angelsolaorbaiceta/InkUnits
Ruby Units	Ruby	2018	26	421	Limited solution	https://github.com/olbrich/ruby-units
SwiftMeasurement	Swift	2017	1	8	Not as comprehensive as other solutions	https://github.com/kenonek/SwiftMeasurement
JNum	Java	2018	1	98	Support for physical units "could be improved" according to the author	https://github.com/attipaci/jnum
MagickScience	C++	2017	1	196	Units of measurement is only a part of this. Not general enough.	https://github.com/Iarfen/MagickScience
Unit.py	Python	2017	1	6	Not as comprehensive as other solutions	https://github.com/fabiano010/Unit.py
Libquantify	C++	2017	1	12	Based on https://www.codeproject.com/Articles/611731/Working-with-Units-and-Amounts	https://github.com/damianorenfer/libquantify
Constants and Units	MatLab	2017	1	4	Not as comprehensive as other solutions	https://github.com/mariomerinomartinez/constants_and_units
UOM-Domain	Java	2018	1	29	A solution specifically related to different domains. Therefore not general enough.	https://github.com/unitsofmeasurement/uom-domain
Units (#9)	C#	2018	1	34	"Under construction"	https://github.com/LabyrinthApps/Units
Scale	Swift	2016	2	25	Not as comprehensive as other solutions, no derived units	https://github.com/onmyway133/Scale
SIConv	Haskell	2018	1	4	Not as comprehensive as other solutions	https://github.com/owainlewis/conventional-si-unit-converter
Convert	Swift	2017	1	8	Not as comprehensive as other solutions	https://github.com/danielbyon/Convert

Converter (#1)	PHP	2017	9	127	Simple solution, lacks support for any derived units.	https://github.com/cartalyst/converter
Quantify	Ruby	2013	5	201	Lacking unit support	https://github.com/spatchcock/quantify
UOM	Ruby	2009	1	10	Not as comprehensive as other solutions	https://github.com/madriska/uom
VUnits	Java	2015	1	15	Only supports limited amount of units	https://github.com/vaslabs/vunits
PhysicalQuantities (#2)	C#	2016	1	9	Lacking documentation	https://github.com/timdetering/PhysicalQuantities
PhysicalQuantities (#5)	Haskell	2017	1	17	Limited solution, no documentation	https://github.com/fehu/PhysicalQuantities
Python-Physical	Python	2015	1	1	Limited solution	https://github.com/EdwinChan/python-physical
PhysicalValue	Swift	2016	1	16	Limited solution	https://github.com/antonvmironov/PhysicalValue
Lathexa Units	JavaScript	2016	1	19	Limited solution, no documentation	https://github.com/lethexa/lethexa-units
Units.jl (#2)	Julia	2014	1	54	Unfinished solution	https://github.com/autocorr/Units.jl
Units (#11)	Go	2016	1	20	Some kind of test project for the developer	https://github.com/zn8nz/units
AutoUnits	C++	2013	1	67	Lacking documentation	https://github.com/Fifty-Nine/AutoUnits
Physical Math	Python	2013	1	39	Limited documentation	https://github.com/matthagy/physmath
Dimanalyser	Java	2013	1	28	Limited solution, no documentation	https://github.com/cymi/dimanalyser
Dftu	C++	2016	1	15	Unfinished	https://github.com/triblatron/dftu
Physcon	Python	2015	1	6	Limited solution, limited documentation	https://github.com/georglind/physcon
Units-v2	C++	2017	1	23	No documentation	https://github.com/Corristo/units-v2
UDUnits	Julia	2018	1	22	A port of UDUNITS2.	https://github.com/Alexander-Barth/UDUnits.jl
Quantity.Net	C#	2017	1	47	Limited solution, no documentation	https://github.com/bcachet/Quantity.Net
UnitManipulationLibrary	C++	2017	1	53	Lacking documentation	https://github.com/OuaisBla/UnitManipulationLibrary

Dimension-TF	Haskell	2012	1	31	Lacks documentation, limited solution	https://github.com/nushio3/dimensional-tf
Phys-Types	C++	2015	1	16	Lacks documentation, limited solution	https://github.com/akubera/phys_types
Unit	C++/CMake/Python	2016	2	25	Lacks documentation, limited solution	https://github.com/njoy/unit
ValueWithUnit	C#	2016	1	22	Old and archived	https://github.com/vbfox/ValueWithUnit
o2scl	C	2018	1	1383	Documentation @ https://isospin.roam.utk.edu/static/o2scl/html/index.html. The unit conversion is just a small(er) part of this.	https://github.com/awsteiner/o2scl
Decibel Units of Measurement with C# Framework	C#	2018	1	5	Built upon an external library (http://sergey-l-gladkiy.narod.ru/index/physics/0-14), limited solution in itself	https://www.codeproject.com/Articles/1236193/Decibel-Units-of-Measurement-with-Csharp-Framework
Quantities, Units, and Values: An Object Oriented Implementation	C	2011	1	8	Only has limited unit support	https://www.codeproject.com/Articles/216191/Quantities-Units-and-Values-an-Object-Oriented-Imp
Automatic Conversion of Physical Units	C#	2014	1		Only has support for 10 units	https://www.codeproject.com/Articles/714545/Automatic-Conversion-of-Physical-Units
TCX Unit Conversion Library	C++	1999	1		Limited solution.	https://www.codeproject.com/Articles/103/TCX-Unit-Conversion-Library
Measurement Unit Conversion Library (#2)	C#	2013	1	3	-	https://www.codeproject.com/Articles/662335/Measurement-Unit-Conversion-Library
International System of Units Notation	C#	2015	1	8	More documentation @ http://www.frank-t-clark.com/Professional/Papers/SI/SI.html	https://www.codeproject.com/Tips/869419/International-System-of-Units-Notation
Converting a value to an SI Unit String	C#	2012	1	5	Limited solution	https://www.codeproject.com/Tips/414254/Converting-a-value-to-an-SI-unit-string

Unit Management Framework	Java	2014	1		Lacking documentation, but keeping anyway	https://sourceforge.net/projects/quantitymanager/?source=directory
UOM Conversion Library	Java	2013	1		No documentation	https://sourceforge.net/projects/uomcl/?source=directory
NumericalChameleon	Java	2017	1		Seems to be a comprehensive solution but lacks documentation.	https://sourceforge.net/projects/numchameleon/?source=directory
Libunits	C	2016	1		Lacking documentation, comments in German.	https://sourceforge.net/projects/libunits/?source=directory
Quantities (#3)	C++	2017	1		No documentation	https://sourceforge.net/projects/quantity/?source=directory
Magnitude (#2)	Java	2013	1		Adaptation of magnitude #1, no documentation	https://sourceforge.net/projects/magnitude/?source=directory
PyPhys Class Library	Python	2016	1		No documentation	https://sourceforge.net/projects/pyphys/?source=directory
Physical	C++	2013	1		"Both compiletime and runtime compatibility checks are available" but lacking documentation	https://sourceforge.net/projects/physical/?source=directory
mcs::units	C++	2013	1		No documentation	https://sourceforge.net/projects/mcs-units/?source=directory
Quan	C++	2016	1		No documentation	https://sourceforge.net/projects/quan/?source=directory
Unum (#2)	Python	2013	1		No documentation	https://sourceforge.net/projects/unum/?source=directory
puny	Python	2013	1		No documentation	https://sourceforge.net/projects/puny/?source=directory
JQuantity: Precision Math Java Framework	Java	2013	1		No documentation	https://sourceforge.net/projects/jquantity/?source=directory
Java library and framework: quantity	Java	2015	1		No documentation	https://sourceforge.net/projects/javaquantity/?source=directory

PhysicalUnits (#2)	C++	2013	1		No documentation	https://sourceforge.net/projects/physicalunits/?source=directory
QUDAL	C++	2012	1		No documentation	https://sourceforge.net/projects/qudal/?source=directory
C++ Unit Library	C++	2014	1		No documentation	https://sourceforge.net/projects/cppunitlibrary/?source=directory
UnitsC++	C++	2015	1		No documentation	https://sourceforge.net/projects/unitscpp/?source=directory
C++ Units	C++	2013	1		No documentation	https://sourceforge.net/projects/cppunits/?source=directory
Unit Var	Python	2013	1		No documentation	https://sourceforge.net/projects/unitvar/?source=directory
Java Measure	Java	2013	1		No documentation	https://sourceforge.net/projects/javameasure/?source=directory
Python Unit Conversion Library	Python	2014	1		No documentation	https://sourceforge.net/projects/pythonconvert/?source=directory
.NET Unit Conversion Library	C#	2013	1		No documentation	https://sourceforge.net/projects/unitcon/?source=directory
.NET Units of Measure	C#	2013	1	1	"Clone of http://www.codeproject.com/Articles/413750/Unit... with several fixes"	https://bitbucket.org/Thecentury/.net-units-of-measure
Measurement.js	JavaScript	2017	2	120	Unfinished project. Seems to be abandoned now.	https://github.com/Philzen/measurement.js
UnitsKit	Objective-C	2016	2	13	Limited solution	https://github.com/stevemoser/UnitsKit
Units-of-Measure (#1)	Scala	2013	1	79	Lacking documentation. Not updated since 2013.	https://github.com/Monomofu/Units-of-Measure
Units (#15)	Rust	2016	1	17	Limited documentation	https://github.com/Boddlnagg/units
FSharp.Units	F#	2017	1	14	Limited solution, but does include units related to "health"	https://github.com/putridparrot/FSharp.Units

Units (#16)	Scala	2016	1	13	Limited solution	https://github.com/nestorpersist/units
Python-Units-Of-Measure	Python	2013	1	6	Limited solution, but good documentation	https://github.com/katerina7479/python-units-of-measure
IMT.Units	Java	2017	1	8	Limited documentation	https://github.com/imt-ag/imt.units-java
PHPMeasures	PHP	2017	2	8	Unfinished project	https://github.com/dcabanaw/phpMeasures
PHP-Units	PHP	2018	2	57	Lacking documentation	https://github.com/hiqdev/php-units
Cuis-Smalltalk-Aconcagua	Smalltalk	2017	1	4	Good documentation but code architecture makes it near unreadable. One file with almost 15,000 lines of code.	https://github.com/hernanwilkinson/Cuis-Smalltalk-Aconcagua
Measurements	PHP	2016	1	17	Lacking unit support (22 units)	https://github.com/marfurt/measurements
UnitsOfMeasureBundle	PHP	2015	1	3	Limited solution	https://github.com/PhpUnitsOfMeasure/UnitsOfMeasureBundle
UnitsOfMeasure (#2)	C#	2013	1	10	Limited solution	https://github.com/capnmidnight/UnitsOfMeasure
Units (#17)	PHP	2018	1	44	Limited solution	https://github.com/ARCANEDEV/Units
Units_of_measure	C++	2016	1	4	Archived	https://github.com/Skeen/units_of_measure
Units (#18)	Go	2018	1	18	No documentation	https://github.com/antha-lang/units
Units of Measure Prototype	Haskell	2014	1	6	Prototype	https://github.com/adamgundry/uom-prototype
Uom.Dart	Dart	2014	2	14	Limited units	https://github.com/damondouglas/uom.dart
RockUnits	C#	2014	1	8	Limited solution	https://github.com/gareth-reid/RockUnits
TundraMeasure.java	Java	2017	1	9	Limited solution	https://github.com/Permafrost/TundraMeasure.java
Units (#19)	PHP	2016	1	100	Solid solution but limited support for different units	https://github.com/marando/Units
UnitOfMeasure (#1)	C#	2016	1	8	No documentation	https://github.com/Chef-Code/UnitOfMeasure

Units-api	PHP	2015	1	21	Limited solution	https://github.com/shrimpza/units-api
cppDegRad	C++	2013	1	7	More documentation @ http://grahampentheny.com/post/cpp11_units_of_measure. Lacking documentation still. And limited solution.	https://github.com/grahamboree/cppDegRad
Unit (#3)	VB .NET	2017	1	11	No documentation	https://github.com/AdamSpeight2008/Unit
Simple.Units	C#	2017	1	38	Limited solution	https://github.com/oriches/Simple.Units
Class-Measure	Perl	2012	1	10	Limited solution	https://github.com/bluefeet/Class-Measure
Measurements	Ruby	2011	1	4	No documentation	https://github.com/Krustal/Measurements
Purescript-Units	PureScript	2015	1	2	Limited solution, limited documentation	https://github.com/fluffynukeit/purescript-units
GenUnits	F#	2016	1	61	Lacking unit support (~30 units)	https://github.com/halcwb/GenUnits
Ruby-Units	Ruby	2012	1	3	Limited solution	https://github.com/davearonson/Ruby-Units
JavaMeasure	Java	2017	1	12	Limited solution	https://github.com/tkuebler/JavaMeasure
AS3Units	ActionScript	2012	1	19	Port of NGUnits	https://github.com/erussell/AS3Units
PHPConverter	PHP	2014	1	42	Not as comprehensive as other solutions	https://github.com/chapmang/PHPConverter
NGX-UOM	JavaScript	2017	1	2	Limited documentation, commits, contributors	https://github.com/catellanir/ngx-uom
Physical Quantities	Smalltalk /C#	2016	2	9	Limited solution	https://github.com/timdetering/PhysicalQuantities
GIM.Quantities	C#	2010	1	34	No documentation	https://github.com/togakangaroo/GIM.Quantities
JUNitConv	Java	2014	2	3	Limited solution	https://github.com/duckler/JUnitConv
Dimensional	Ruby	2010	1	47	Lacking documentation	https://github.com/cch1/Dimensional
UnitOfMeasure (#2)	Scala	2014	1	12	Older solution, limited contributors, limited commits	https://github.com/ricemery/unitofmeasure

Umpy	Python	2016	1	8	Limited solution	https://github.com/perdixsw/umpy
SBUnits	Swift	2017	1	13	Limited contributors, limited commits	https://github.com/EBGToo/SBUnits
Units (#21)	Haskell	2018	8	308	Specifically related to meterology	https://github.com/goldfirere/units
DDUnitConverter	Objective-C	2016	1	22	Limited unit support	https://github.com/davedelong/DDUnitConverter
PHP-Conversion	PHP	2016	7	95	Limited documentation	https://github.com/crisu83/php-conversion
SIUnitX	TeX	2018	4	2165	Lacking documentation	https://github.com/josephwright/siunitx
Units (#24)	Scala	2016	2	111	Pre-release ("experimental") state, not updated since 2016.	https://github.com/KarolS/units
SI-Units	Java	2018	7	242	Limited unit support (weight, length, volume)	https://github.com/unitsofmeasurement/si-units
Conversion (#1)	PHP	2017	6	29	Limited unit support	https://github.com/abhimanyu003/conversion
UnitConverter (#5)	Python	2018	14	99	Lacking documentation	https://github.com/mattgd/UnitConverter
Unit-converter	PHP	2018	1	165	Limited unit support	https://github.com/jordanbrauer/unit-converter
HHUnitConverter	Objective-C	2015	2	17	Limited solution	https://github.com/HiveHicks/HHUnitConverter
Angular-Unit-Converter	JavaScript	2016	2	18	Limited solution	https://github.com/alexandernst/angular-unit-converter
UnitConverterWin	C	2016	2	368	In Russian	https://github.com/NikolaiTyrMDS/UnitConverterWin
UnitConversion (#1)	Objective-C	2011	1	9	Limited documentation	https://github.com/unixpickle/UnitConversion
Cropio_Units_Converter	C++	2016	1	8	Limited solution	https://github.com/cropio/cropio_units_converter
Quantity	Ruby	2011	3	85	Only supports weight	https://github.com/bhuga/quantity
UDUNITS-2	C	2018	5	638	Lacking documentation	https://github.com/Unidata/UDUNITS-2
Quantities	Swift	2016	1	10	Limited solution	https://github.com/BradLarson/Quantities

PhysicalQuantities	Python	2018	1	241	Lacking documentation	https://github.com/juhasch/PhysicalQuantities
Quantities	R	2018	1	28	No documentation	https://github.com/r-quantities/quantities
ScientificQuantities	C++	2017	2	41	Limited documentation	https://github.com/nourani/ScientificQuantities
Physical-Quantities	Python	2016	1	6	Lacking documentation	https://github.com/hplgit/physical-quantities
ScalaQuantity	Scala	2012	1	32	Limited contributors, limited commits	https://github.com/zzorn/ScalaQuantity
Quantities	C++	2013	1	35	Written as an exercise	https://github.com/djbarker/quantities
Quantity	Ruby	2009	1	37	Limited solution	https://github.com/aportnov/quantity
Quantities	Batchfile	2016	1	6	No code? Just documentation?	https://github.com/greatfriends/Quantities
Quantities	C#	2017	1	13	"This is primarily a project that lets me explore how far one can push C#'s generics"	https://github.com/atmoos/Quantities
Physics-Measurement	JavaScript	2015	1	15	Limited solution	https://github.com/victorpotasso/physics-measurement
PyQuantity	Python	2015	1	6	Limited contributors, limited commits	https://github.com/gabbpuy/PyQuantity
Sundew.Quantities	C#	2017	1	13	Limited solution, no documentation	https://github.com/hugener/Sundew.Quantities
Meteor-Quantities	JavaScript	2018	1	14	Limited contributors, limited commits. Specific domain.	https://github.com/luzlab/meteor-quantities
Quantity	Clojure	2015	1	31	Limited solution. Refered to as a "work in progress" but abandoned.	https://github.com/spellman/quantity
Convert-Quantities	JavaScript	2017	1	5	No documentation, but still good enough to keep in	https://github.com/kendru/convert-quantities
UnitConversion (#2)	C#	2018	3	91	Limited solution	https://github.com/gkampolis/UnitConversion
Convert.js	JavaScript	2018	9	240	Lacking implementation documentation. Referred to as "simple" by author.	https://github.com/YazilimMuhendisiyizBiz/convert.js

MeasureBundle	PHP	2018	5	18	Interesting solutioin (using families?) but not tier 1	https://github.com/akeneo/MeasureBundle
Convertr	R	2017	3	76	Lacking documentation	https://github.com/ropenscilabs/convertr
Metric	Java	2014	2	10	Project "just starting"	https://github.com/gnapse/metric
Jsunitconverter	JavaScript	2016	1	11	Limited solution	https://github.com/jerodvenemafm/jsunitconverter
Sius	Java	2015	1	79	Lacking unit support	https://github.com/mbe24/sius
Unit	JavaScript	2018	5	47	Lacking documentation, lacking unit support (15 units)	https://github.com/smartcar/unit
Unit-formula	Common Lisp	2017	3	64	Limited solution	https://github.com/Ramarren/unit-formula
Node-units	JavaScript	2016	1	17	Limited solution	https://github.com/brettlangdon/node-units
Units.js	JavaScript	2012	1	5	Limited contributors, limited commits	https://github.com/jairtrejo/units.js
Units (#28)	C++	2015	2	9	Limited documentation	https://github.com/lonkamikaze/units
Unit.styl	JavaScript	2015	1	29	Limited solution	https://github.com/diessica/unit.styl
Conversionr	R	2017	3	39	Only supports some units	https://github.com/alexnakagawa/conversionr
Math-units	Perl	2015	2	13	Limited solution	https://github.com/kenfox/Math-Units
UnitConvert	JavaScript	2015	1	10	Limited solution	https://github.com/agnoster/unitology
NGunits	Java	2011	2	9	Limited solution, specifically related to runtime approach	https://github.com/erussell/ngunits
Units (#29)	Swift	2015	1	6	Limited solution	https://github.com/chrisjeane/Units
UnitConversion (#3)	Python	2015	3	424	Lacking documentation	https://github.com/UnitConversion/unitConversion
Metiri	JavaScript	2015	1	69	Limited solution	https://github.com/GCheung55/metiri
Convert-units	Ruby	2018	1	173	Limited unit support	https://github.com/tanviroo2700/convert_unit

Django-Unit-Field	Python	2015	1	49	Unfinished	https://github.com/kohout/django-unit-field
Units (#30)	PHP	2013	1	7	Uses a graph model	https://github.com/rmasters/units
Unit_Conversion	Ruby	2015	1	25	Limited solution	https://github.com/eternal44/unit_conversion
Maegor	JavaScript	2017	1	67	Lacks support for arithmetic operations	https://github.com/lukaskollmer/maegor
Units-System	Ruby	2012	1	76	"Experimental"	https://github.com/jgoizueta/units-system
Physical	C++	2018	2	144	Limited documentation	https://github.com/olsonse/physical
Conversion (#6)	C++	2016	1	1	Limited solution, limited documentation	https://github.com/simonmeaden/conversion
Gorilla	Ruby	2011	1	36	Limited solution	https://github.com/stephencelis/gorilla
QtUnits	C++	2018	1	44	Limited solution	https://github.com/hrobeers/QtUnits
Units (#32)	Ruby	2009	1	1	Limited solution	https://github.com/woahdae/units
convertR	R	2016	1	10	Incomplete solution	https://github.com/colin-fraser/convertR
UnitConversion (#4)	C#	2017	1	33	Only supports length, mass, volume, area.	https://github.com/Mecteral/UnitConversion
JSR 275	Java	2018	1	14	No documentation	https://github.com/unitsofmeasurement/jsr-275/tree/master/src/main/java
Quantified	Ruby	2015	5	14	Only has length + mass	https://github.com/Shopify/quantified
Units (#33)	Haskell	2017	1		Domain specific	https://hackage.haskell.org/package/units