The Symbiotic Nature of Requirements Quality and Software Cost Estimation

By Colin Hammond, January 2023.

Abstract
It is widely recognised that high quality software requirements lead to reliable functional size estimates, what is less apparent is the potential contribution of functional sizing to help assess and improve the quality of requirements. This paper looks at the observations from assessments of software user stories using our NLP (Natural Language Processing) requirements analysis tool, and how this symbiotic relationship is revealed and enhanced with automated analysis tooling.

Introduction to Software Cost Estimates and Functional Size

Executives seek answers to the questions "how much will it cost?" and "how long will it take?" Valid estimates enable executives to make sound decisions about planning software work and software acquisitions. Poor cost and schedule estimates can lead to poor and costly business decisions. Hence the valuable role of cost estimators.

In software work, the main cost component is human effort. That effort involves many activities, principally: the effort to define, design, develop, test, fix, and deploy a software system.

Obtaining effort or cost estimates by directly asking the developers or experts for their opinion is notoriously unreliable. Some of the reasons for this unreliability are:

- failing to adopt a universal sizing approach,
- a tendency to focus on the initial development work only,
- a tendency to overlook scope that has not been presented but is implicit, "knowable unknowns",
- human biases, a tendency to underestimate cost to "win the work" (in rare cases the opposite may be true).
- a tendency to not factor in delays caused by external factors and unknowns,
- a tendency to overlook the rework associated with changing requirements

To overcome some of these problems, we use a standardised means of sizing software, functional sizing. Functional sizing allows us to measure the size of a piece of software, irrespective of the technology used to create it. It focuses on sizing only user-recognisable functionality. There is strong, proven, correlation between the functional size of a piece of software and the effort to perform the activities involved in creating the software. And so, functional size is a proven and excellent basis for effort estimates.

There are two leading standards for functional sizing. They are the IFPUG Function Points (FP) and the COSMIC Function Points (CFP).

Functional size can be measured from user requirements or from the code once it has been written. The former is most valuable as it allows for sound decisions to be made before committing time and resources.

The findings in this paper are based on over a hundred assessments of sets of software requirements for customers using our requirements analyser tool.

Hereafter when I use the terms "user story" or "requirement", I am referring to a functional user requirement. When I use the term estimate I usually mean an estimate range.

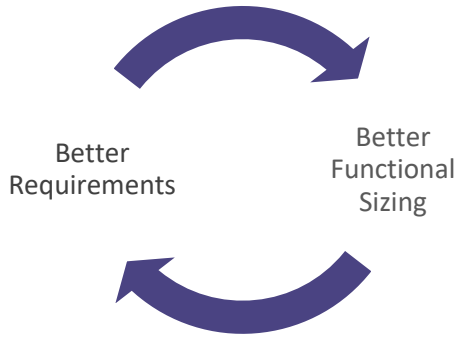
Early Estimating of Functional Size

For a functional size measurement, we must know ALL the user requirements in detail and the logical data design of the solution. It is both rare and unreasonable to expect that all the requirements can be known at the start of a project.

The usual goal of cost estimator is to obtain a reliable cost estimate at a time when the user requirements and the design are not yet known. Rather than measure, he must estimate functional size, using all available information. Most of the information he uses, the clues to the real size, are obtained from written user requirements, data models and other documentation. This is where the relationship between the quality of requirements and the quality of size estimates is self-evident.

How requirements quality helps functional sizing and vice versa

Not only do better requirements lead to better estimates, but what is not always recognised is that **the process of functional sizing can help to improve the quality of the requirements.** The functional sizing exercise is an effective means of verifying the interpretation of requirements and thus for improving requirements quality.



Where the requirements are ambiguous (unclear functionality), incomplete, inconsistent, or just wrong, the cost estimator must seek clarifications/corrections from the BA's, product owners or other stakeholders. The estimator is clarifying and discovering the real requirements, whether they are as written, or not. They are verifying what is known and

exposing what is unknown. If these clarification challenges are used constructively by BAs, then requirements quality can be improved as a by-product of functional sizing.

There is an opportunity here to solve two problems simultaneously - improve estimates and improve requirements quality.

Of the unknowns in the scope, some are knowable and some are unknowable.

Classic examples of knowable unknowns are the CRUD complements to a single transaction process for an object. E.g. suppose only one user story mentions *list products*, therefore there must be other transactions (knowable unknowns) for the create, update and delete of *products*.

Other examples of knowable unknowns would be the splitting of Epics into component user stories.

Other unknowns may not be knowable, such as a change in business model that leads to changes in requirements. The cost estimator should seek to **minimise the unknowables**, by exposing the knowable unknowns as much as possible. They can then make a cost allowance for the remaining unknowable unknowns.

The main observed requirements quality attributes that impact sizing

There is extensive literature on what is meant by requirements quality, here is our preferred list of 10 quality attributes for software requirements. (Nb this list addresses the main quality attributes of application software user stories, or functional user requirements. It should not be considered as a quality checklist for system design specifications):

1. Clear *
2. Concise *
3. User-oriented *
4. Testable *
5. Measurable *
6. Consistent *
7. Complete *
8. Unique *
9. Valuable
10. Design-free

*Many of these requirements' attributes are examined and challenged during the exercise of functional sizing.

Granularity, Completeness and Sizing

Requirements granularity impacts size estimation, and is a leading indicator of requirements' completeness.

Software requirements generally follow a hierarchy such as:

- **Objectives**
- **Epic / Capability**
- **User Story / Feature level**
- Business rules / constraints
- Detailed, attribute-level acceptance criteria

For early functional sizing we are generally working with Objectives, Epics and User story of coarse granularity which are more likely to be incomplete, which leads to understated size estimates.

Let's look at two examples:

`As a sales agent want to be able to edit a contact's profile`
(3 CFP)

This is a typical early, high-level requirement. Whilst it is sizeable, it is unlikely to be complete. A more complete version might of the same requirement might be:

`As a sales agent, first verify that I have permissions to the profile, then verify that the contact profile is not locked, then I can edit the contact's profile,`
(9 CFP).

#1 and #2 are the same story, but #2 is just more detailed, and is 3x larger, 9 CFP vs 3 CFP. When dealing with a large set of requirements, the impact of granularity on estimates can be dramatic. Estimates based on course granularity will be incorrectly low.

Now let us look at a more extreme example, but not an uncommon challenge, for cost estimators:

`As a sales agent I want the contact profiles synchronised between the CRM, mailing system and customer app.`
(40+ CFP)

This is a high-level description of a capability (or an epic) that needs to be broken down further into constituent user stories (story slicing) for us to size accurately. In fact functional sizing is a good test for the need for story slicing. From the perspective of requirements quality: coarse granularity, such as this, can be considered as incomplete and untestable. Notwithstanding an experienced estimator can still estimate a size to this type of functionality (as we have done at 40CFP+). Many user stories that we see have no functional size, our observed range is a non-normal distribution of 0 - 90 CFP.

This example highlights why the practice of counting user stories, or counting requirements is an unsafe basis for cost estimation.

Observed Sizes

Automated functional sizing of requirements depends entirely on the language used by the requirements author(s). Poor quality requirements tend to be the norm and consequently the initial size estimates are less accurate. The tooling can create a fast feedback loop between requirement refinement and sizing accuracy, which means that better estimates and better requirements can be achieved faster.

For requirements where functionality has been detected, the average size is around 4 CFP. After refinement, we see slightly higher average story size of 5-7 CFP.

Clarity / Ambiguity

If the actor and the functional intent are not clearly evident from the language of the user story, then it can be considered unclear, or ambiguous. Sometimes requirements are not intended to be functional, they should be categorised accordingly and be subject to a suitable cost estimation approach.

Unclear Functional Intent

Some requirements that are meant to be functional, cannot be sized because of unclear language used by the requirements author. For example:

```
I want to know the sales by group to decide on future budget allocation.
```

The functionality intended by this requirement is highly likely to be misinterpreted by readers. We don't know from these words, who is the user, nor what data groups (objects) are being accessed and how. This is an example of lazy work by requirements authors, it is a poor requirement that cannot be reliably sized.

Resolving the problem is usually straightforward when cost estimators ask these three questions:

1. *Who is the user?*
2. *what data groups that are being moved?*
3. *what type of data movements are they?*

Not functional requirements

Technical tasks and *Non-functional requirements (NFRs)* cannot be sized in the same way as functional requirements, and yet, they are commonly mixed in (a backlog) with functional requirements. Both technical tasks and NFRs require cost estimation, but the approach to estimating the costs of these two groups is different from sizing the functional aspects of the software. Detection and correct categorisation is important for the cost estimators. Automated detection of NFRs and functionality can help the sizer categorise these more rapidly.

Completeness

These are the main aspects of *requirement completeness* that impact functional sizing:

Completeness across a set of requirements (CRUD completeness)

For a software system to be complete, any data stored within it should be fully maintained. Each data type (ILF in IFPUG or OOI in COSMIC) will usually require a full set of CRUD functions for persisted data. There should be at least one requirement that creates the data of that type and one for each of read, update and delete. The work of identifying and cross-referencing CRUD functions from requirements is called CRUD analysis and involves maintaining a CRUD matrix. Traditionally CRUD analysis is very labour intensive, and consequently tends not to happen.

CRUD analysis exposes knowable unknowns.

In our observations, early requirements tend to be woefully incomplete in this respect. Typically, we have seen that early requirements are missing 20-60% of essential CRUD events on detected objects (see also consistency of object naming). This means that the actual functional size can be 2x the initial estimate, an underestimate like this can turn a viable project into a non-viable one. With automated CRUD analysis this is revealed immediately. As in the example below, missing requirements tend to outnumber duplicates between 5 and 10:1.

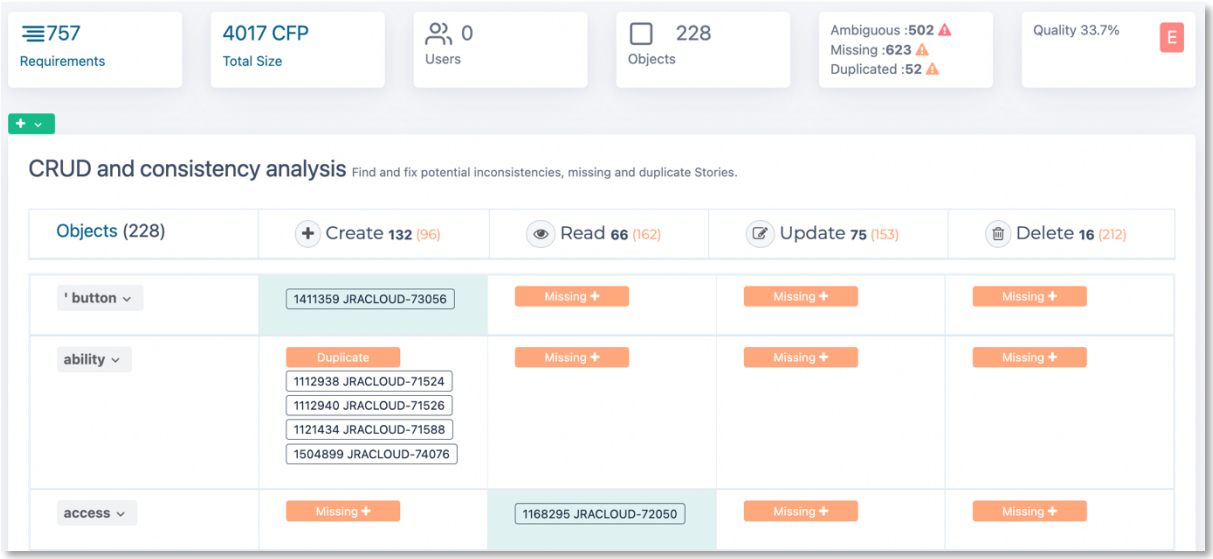


Figure 1 Automated CRUD analysis reveals missing and duplicated functionality.

Internally Completeness of a Requirement

The second aspect of completeness is whether each requirement is complete within itself. To be complete, all the functional steps or transaction processes, should be described. For example:

An incomplete example*:*

```
When I login, check that my username and password match.
(3 CFP)
```

A more complete version of the same requirement*:*

1. As a registered user, when I login, first search for my profile. (3 CFP)
2. Compare my profile against the blacklist (3 CFP)
3. Look up the organisation in which I am a member and verify that the organisation is not blacklisted. (3 CFP)
4. Update my profile with last_login time (3 CFP)
5. Look up my permissions to perform work then (3 CFP)
6. Display my welcome screen (3 CFP)

Total 18 CFP

The refined example is 18 CFP, 6 times larger than the unrefined version. A more complete requirement is higher quality and leads to a more reliable estimate.

Buried Functionality

The third consideration of requirement completeness is that sometimes the user story itself is a pithy statement rather like the unrefined example above, but on closer examination of the acceptance criteria and other notes that accompany the story, we discover additional, hidden functionality. It is important from a sizing and design perspective to expose all of this functionality and not to hide it in the acceptance criteria (which are principally used for test design). NLP can help us discover this hidden functionality very quickly. Making the job of the cost estimator much easier.

Consistency of Terms

When looking at a set of requirements we should use consistent terms for **Users** (personas) and **Object types** (ILFs). Inconsistent terminology will lead to them being misinterpreted as distinct and consequently size estimates might be inadvertently higher than they should.

In particular, inconsistent use of terms for objects can have a significant impact on size estimation. For example, you might encounter several terms for the same thing across a set of requirements such as: *categories*, *category name*, *product categories.*

Unless you resolve this inconsistency, your size estimate could be multiple times bigger than it should be. The size sensitivity is more pronounced with IFPUG than with COSMIC because IFPUG requires an extra data function of 7-15 FP in addition to the transaction functions associated with maintain and moving the data.

Larger projects tend to have more people involved in writing requirements which leads to even less consistency in terminology. Automation and NLP can help us to spot this problem rapidly so that it can be resolved quickly and early. This is a key area where functional sizing and requirements quality are interdependent.

Discovering real requirements through Modelling

Other approaches to requirements modelling can help to expose the real requirements and the eventual functional size. Visually modelled requirements are particularly helpful for spotting outliers, mistakes, and inconsistencies across a set of requirements. The diagram below has been generated from the NLP analysis. This can be used interactively to help improve requirements quality and size estimates simultaneously.

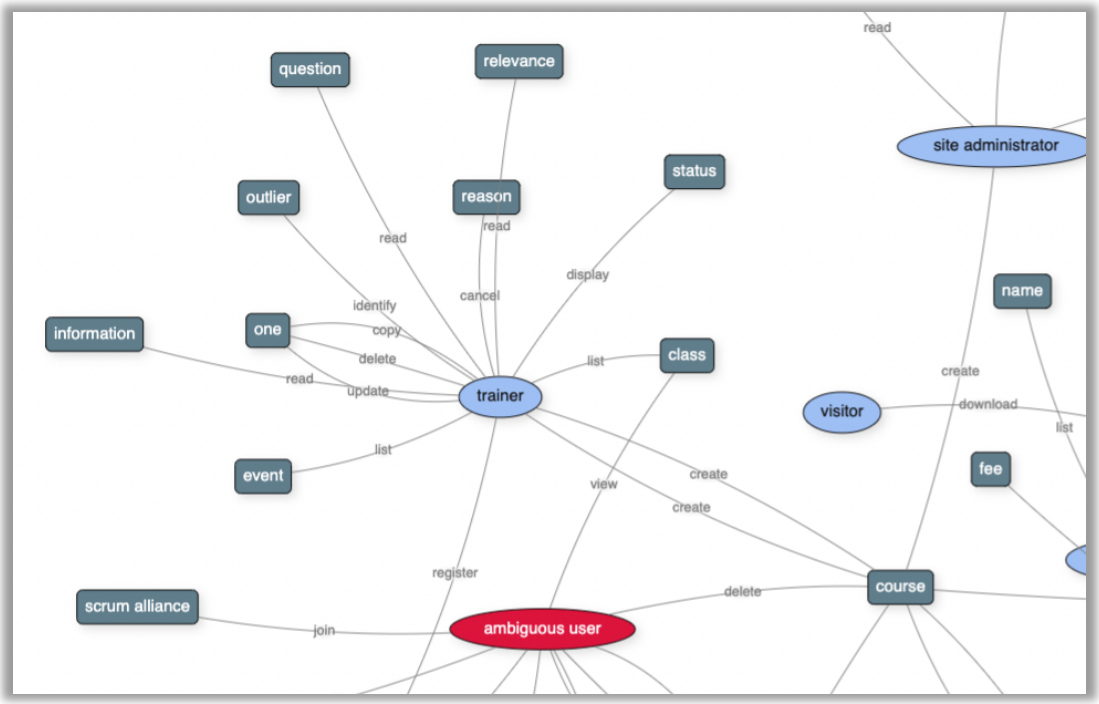


Figure 2 Dynamic modelling using NLP on user stories, gives insight that can help cost estimators.

This diagram is a partial screenshot of a complete set of user stories that have been auto-analysed. It shows all the actors and functionality detected across the set of user stories. Where it says "ambiguous user", these are user stories, where the language describes functionality but no actor was detected. This diagram also helps us discover and validate whether we have all the correct requirements for each persona. This can also help us to consider the applicability of some NFRs too.

Summary of the Requirements Quality Characteristics on Sizing

Overall, our observations from automated analysis have revealed the following observations about the relationships between functional sizing and requirements quality (in no particular order):

Requirements Quality Attribute	Affects	Indicative Range: actual vs initial estimate	Explanation	Typical observation
Functional completeness of a requirement	Requirement	0 to +200%	Functionality is often omitted or buried in acceptance criteria. Actual can be 4x larger than initial size estimate.	Most requirements understate functionality and end up 50% - 100% bigger than initially stated/estimated.

CRUD Completeness across a set of reqs.	Set	-20% to +400%	We sometimes see only one function mentioned when a full set of CRUD is required. Duplicates are far less comon.	Most sets only include only 50% of required CRUDs.
Missing requirements, revealed through modelling. (inc user oriented).	Set	0 to 50%	Manual or automated modelling can expose "hard-to-reach knowable uknowns".	Typically, 10- 30% of missing functionality can be detected this way.
Ambiguous functional requirements	Requirement	0 to +300%	Functional ambiguities due to poor language use, often mask understatements of scope.	About 40% of all requirements are initially unsizable. Using a tool like ScopeMaster from the outset eliminates this problem very quickly.
Object Naming Inconsistency	Set	-150% to +20%	Inconsistent object names can lead to overestimate of size. (*The only item in the table that leads to early overestimation.)*	This is common and tends to overstate initial automated size detection.
Methodology Choice: IFPUG vs COSMIC vs SFP	Set	-30% to +30%	The gross FP count discrepancy between these methodologies is less significant than other factors.	Automated IFPUG estimates are governed by the ILF complexity assessment which is very hard with NLP. COSMIC does not suffer this variability.
Accuracy of measurement	Set	-15% to +15%	Whether using automation or sizing manually, rarely more than 15% variance for CFP.	A formal test has shown automation in CFP to be within 15% of a manual count.
NFRs that are actually functional	Set	0-30%	When an NFR is assumed to be not functional but actually is.	functional security requirements are often overlooked.

From this we can observe that it is important for a cost estimator, when performing early functional sizing, to pay close attention to the following:

1. Individual requirement clarity
2. Individual requirement completeness
3. Consistency of terms for object types

4. Requirements set completeness
5. Hidden functionality that can be revealed using modelling.
6. Hidden functionality that is buried in acceptance criteria (ie. no CRUD should be in the acceptance criteria).

Non-Functional Requirements

NFRs are not the primary focus of the functional sizer, but they are a concern to the cost estimator as some NFRs can carry a significant cost to satisfy. It is worth considering the following differences that apply to NFRs:

1. Some NFRs are functional. Notably, security requirements are usually functional although often misclassed as NFRs. Elaborating and refining these may be appropriate as part of sizing.
2. NFRs should be quantified so that acceptance can be tested. Some NFRs can have a significant impact on cost. Early detection of system-wide NFRs can impact architecture and high level design, so need to be detected early.*
3. Some NFRs that affect only parts of a system can make use of function size of those parts of the system to indicate potential cost.

*Automation can help to expose language that infers specific types of NFR, and in so doing, can produce a matrix of detected and missing NFRs. This helps us to ask the right questions early to anticipate and contain the risks of NFRs surprising us later. **This is another example of how the cost estimator can help contribute to better requirements by exposing knowable unknowns.**

About Requirements Quality and Functional Sizing in Practice

Poor requirements quality and the wasteful consequences of poor requirements are prevalent throughout the software industry. The specific wasted work caused by poor requirements is rarely quantified objectively. It is sometimes classified by teams as "refactoring in response to evolving requirements". Nonetheless, with better initial requirements, it is avoidable waste.

Requirements quality assurance is rarely an assigned responsibility on software projects, except in systems engineering work. Accenture reports that as much as 35% of all defects found in larger production software solutions are caused by poor or incomplete requirements, and yet little is done to address the root cause of the problem.

The most common estimation approaches being used are story counts, story points or T-shirt sizing, all of which are profoundly inferior to functional sizing. Even though we have shown that functional sizing is so beneficial, both for estimation and for improving requirements, it is not widely used, yet. Perhaps with the automation of Simple Function Points and COSMIC function points, this will change in the coming years.

When reviewing requirements for Agile teams, we constantly see that user stories are of poor quality and incomplete. They are known to cause waste and rework. And yet many

agile teams resist analysis by offering the glib justification of “we are discovering the requirements as we go, no need for wasteful analysis”. I also believe that resistance to using functional sizing is really the fact that agile teams enjoy the fact that their work is not measured.

Automation of Sizing and Requirements QA

Automated requirements analysis tools can help significantly with both sizing and requirements QA. With a typical starting position of 1 defect per FP in requirements, automated tools can now expose 50% of these effortlessly.

For large projects, say 1000 function points, the learning from the automated analysis of the user stories for the initial 100 function points dramatically decreases the defect potential in the user stories for the remaining 900 function points.

Unless your organisation benefits from poor requirements and an absence of functional sizing, and some do(!), you can automate sizing and requirements analysis simultaneously.

For example, CRUD analysis on a set of 500 requirements might typically involve 1000 CRUD functions overall. Manually maintaining a CRUD matrix of this size would require one or two fulltime staff, and is therefore unlikely to happen. The consequences of which are that the teams are left to work with inferior requirements. Rework levels will be higher and cost estimates will be less reliable.

Similar benefits arise from other automated capabilities such as: user consistency detection, object consistency detection, NFR detection, automated use case modelling and ambiguity detection, class modelling. When the tooling makes these proven analysis techniques effortless, it unlocks the ability for teams to achieve high quality and more complete requirements much earlier in the project lifecycle.

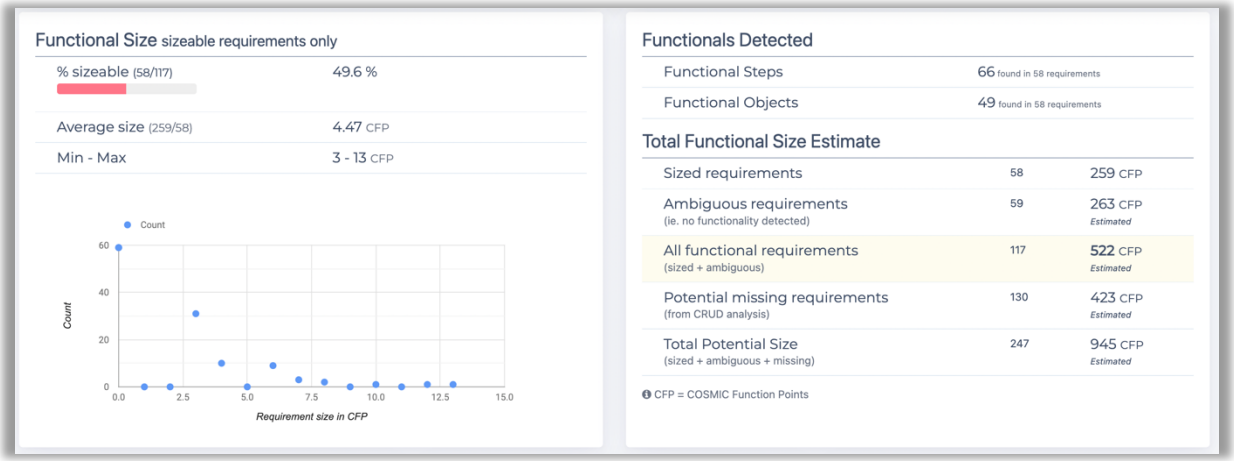


Figure 3 Automated sizing can help the cost estimator provide more dependable estimates, by factoring in requirements quality.

Conclusion

Functional sizing is an excellent technique for creating reliable cost estimates for software work. Early functional size estimation requires probing into the meaning of functional user requirements, and as by product, the requirements quality can be assessed. Problems within the requirements are revealed (and potentially fixed). Functional sizing can actually be an excellent technique specifically for testing and improving requirements.

The impact of requirements quality on early functional size estimation is non-trivial. Cost estimators must consider the various aspects of requirements quality when preparing estimates based on early requirements especially functional clarity, completeness, and consistency of object naming.

In order to size a system of about 1,000 function points (as much as $5m of government project), it only takes about 200 user stories, about 2400 words. This is often only 3 or 4 weeks work, especially using automated analysis and refinement. This provides the foundation for costs/fp, fp/day and defects/fp. The product owner or BA will still have plenty of detailed work to support the developers, **but the size of the scope can be known quickly and with confidence**.

The activity of determining functional size can lead to improved requirements quality and vice versa, hence the symbiotic relationship.

Poor estimates lead to poor and costly management decisions. Poor software requirements lead to high levels of rework and waste within the team(s). Functional sizing can simultaneously help to solve both of these problems. It is therefore a profoundly valuable activity. NLP and automated analysis tooling makes both sizing and requirements analysis faster and more thorough than ever.

Often, the commercial model for software development contracts benefits the vendors when the requirements are poor quality and the adoption of proper functional sizing throughout a project further harms their commercials.

To achieve the combined benefits of better estimates and better user stories, it is up to cost estimators and buyers of software development contracts to encourage the adoption of functional sizing and the simultaneous improvement in requirements quality. This will lead to more transparency, better estimates and less waste, and the final goal of delivering software faster, better and cheaper.