Measuring Software Engineering Report

Measurable Data - Collecting Software Engineering Data

The operational activities of measurement begin with collecting and retaining data. The right people, sensors, tools, and practices into the processes must now be put in the right places. This also means capturing and storing the data for subsequent use in analysis and process improvement.

The principal tasks associated with collecting and retaining data for process management are:

- Design the methods and obtain the tools that will be used to support data collection and retention.
- Obtain and train the staff that will execute the data collection procedures.
- Capture and record the data for each process that is targeted for measurement.
- Use defined forms and formats to supply the collected data to the individuals and groups who perform analyses.
- Monitor the execution and performance of the activities for collecting and retaining data.

The challenge of collecting software engineering data is to make sure that the collected data can provide useful information for project, process, and quality management and, at the same time, that the data collection process will not be a burden on development teams. Therefore, it is important to consider carefully what data to collect. The data must be based on well-defined metrics and models, which are used to drive improvements. Therefore, the goals of the data collection should be established and the questions of interest should be defined before any data is collected. Data classification schemes to be used and the level of precision must be carefully specified. The collection form or template and data fields should be pretested. The amount of data to be collected and the number of metrics to be used don't

need to be overwhelming. It is more important that the information extracted from the data is focused, accurate, and useful rather than plentiful. Without being metrics driven, over-collection of data could be wasteful. Overcollection of data is quite common when people start to measure software without an a priori specification of purpose, objectives, profound versus trivial issues, and metrics and models.

Gathering software engineering data can be expensive, especially if it is done as part of a research program, for example, the NASA Software Engineering Laboratory spent about 15% of their development costs on gathering and processing data on hundreds of metrics for several projects (Shooman, 1983). For large commercial development organizations, the relative cost of data gathering and processing should be much lower because of economy of scale and fewer metrics. However, the cost of data collection will never be insignificant. Nonetheless, data collection and analysis, which yields intelligence about the project and the development process, is vital for business success. Indeed, in many organizations, a tracking and data collection system is often an integral part of the software configuration or the project management system, without which the chance of success of large and complex projects will be reduced.

In their study of NASA's Software Engineering Laboratory projects, Basili and Weiss (1984) found that software data are error-prone and that special validation provisions are generally needed. Validation should be performed concurrently with software development and data collection, based on interviews with those people supplying the data. In cases where data collection is part of the configuration control process and automated tools are available, data validation routines (e.g., consistency check, range limits, conditional entries, etc.) should be an integral part of the tools. Furthermore, training, clear guidelines and instructions, and an

understanding of how the data are used by people who enter or collect the data enhance data accuracy significantly.

The actual collection process can take several basic formats such as reporting forms, interviews, and automatic collection using the computer system. For data collection to be efficient and effective, it should be merged with the configuration management or change control system. This is the case in most large development organizations. For example, at IBM Rochester the change control system covers the entire development process, and online tools are used for plan change control, development items and changes, integration, and change control after integration (defect fixes). The tools capture data relevant to schedule, resource, and project status, as well as quality indicators. In general, change control is more prevalent after the code is integrated. This is one of the reasons that in many organizations defect data are usually available for the testing phases but not for the design and coding phases.

An Overview of the Computational Platforms Available to Perform this Work

Increasingly, researchers are turning to computational models to understand the interplay of important variables on systems' behaviours. Although researchers may develop models that meet the needs of their investigation, application limitations—such as nonintuitive user interface features and data input specifications—may limit the sharing of these tools with other research groups. By removing these barriers, other research groups that perform related work can leverage these work products to expedite their own investigations. The use of software engineering practices can enable managed application production and shared research artifacts among multiple research groups by promoting consistent models, reducing redundant effort, encouraging rigorous peer review, and facilitating research collaborations that are supported by a common toolset.

Computer modelling (i.e., simulation) is increasingly being used to understand systems that either are too complex to have closed-form analytic solutions or are dynamic systems in which the behaviour of a system changes over time. Increasing model complexity has fostered the sharing and joint development of models by multiple, independent research groups to leverage these work products and expedite investigations.

The development and evolution of computer models involves dynamic model expansion and modification to address new scientific questions; therefore, effectively sharing models among investigators can be a challenge. Because model-specific limitations, such as nonintuitive user interface features and data input specifications, may limit ease of use by researchers who are unfamiliar with an application, models created for a specific research study are rarely used outside the originating research group.

Standard software engineering practices that researchers can adopt during the model development process to increase their usability by external research groups are described.

General Software Development Tasks

The software development process should follow a standard product development method. Several recognized software engineering practices can benefit the development and implementation of computational models by promoting standard development procedures that encompass the development process from design to deployment. The use of standard documentation tools in particular can facilitate wider model distribution among research groups.

In general, the software development process life cycle typically has five phases:

- **Requirements phase**: Gather information detailing the requisite features and use cases of the software product based on the problem statement.
- **Design phase**: Extract implementation constructs from the requirements documentation.
- **Implementation phase**: Implement constructs identified in the design phase in the target programming language.
- **Testing phase**: Execute the software product to ensure that it exhibits the proper behaviour as specified in the requirements.
- **Deployment phase**: Transition the successfully tested product to its production environment.

By using software engineering practices that emphasize modular design and implementation to allow for software reuse, extensibility, and complexity management, researchers can develop systems that are easier to maintain internally. They can also develop more robust and usable computational modelling solutions that are useful beyond the originating research group, thereby supporting collaborative efforts. Employing well-established and accepted industry analysis and implementation practices can facilitate this type of model development: three important practices are use of an iterative software development process (ISDP), which I have outlined below, object-oriented methodology (OOM), and Unified Modelling Language (UML).

Software Engineering Practices

Waterfall Software Development Process: A well- known software development process is the waterfall process (Figure 1). In this, developers perform the five software development phases sequentially, with the output of one phase serving as input to the next phase. All input information for a specific phase is available to researchers while performing activities within that phase; however, once completed, developers do not revisit previous phases.

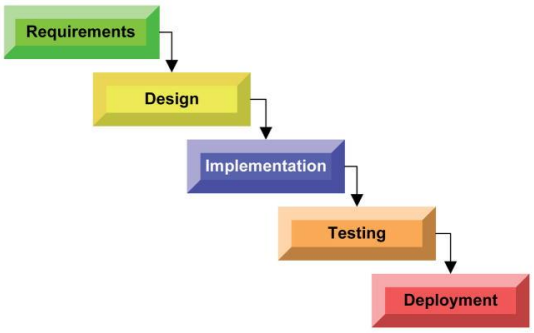


Figure 1: Waterfall software development process

This is, however, an idealized model. In reality, not all the information needed for a specific phase may immediately be known or available; moreover, an application's parameters may change as a result of a phase's outputs. Thus, the waterfall process does not necessarily support the realities of actual software development and may not be the most efficient process for certain projects.

Iterative Software Development Process

In contrast to the waterfall process, the iterative software development process (ISDP, see Figure 2) allows for flexibility during software development. In this process, developers conduct multiple cycles of the waterfall process before delivering the final software product. At the conclusion of each cycle, developers produce an interim software product that incorporates a subset of the requirements to be tested and evaluated. Each interim product serves as additional input for the next development cycle. Thus, developers can revisit the development phases; this capability allows for sufficient review, management, and incorporation of any needed software changes and yields a product that evolves over time using multiple evaluations.

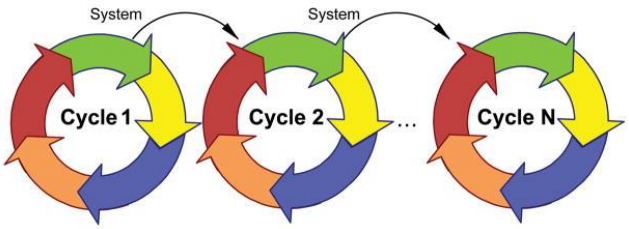
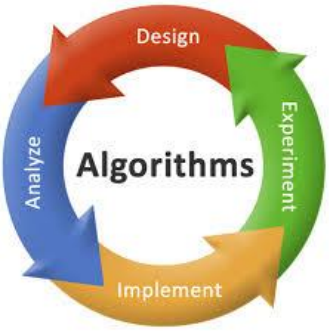


Figure 2: Iterative software development process

Algorithmic Approaches Available

Algorithmic efficiency is a property of an algorithm which relates to the number of computational resources used by the algorithm. An algorithm must be analysed to determine its resource usage. Algorithmic efficiency can be thought of as analogous to engineering productivity for a repeating or continuous process.

For maximum efficiency, we wish to minimize resource usage. However, the various resources (e.g. time, space) cannot be compared directly, so which of two algorithms is considered to be more efficient often depends on which measure of efficiency is considered the most important, e.g. the requirement for high speed, minimum memory usage or some other measure of performance.



The importance of efficiency with respect to time was emphasised by Ada Lovelace in 1843.

Early electronic computers were severely limited both by the speed of operations and the amount of memory available. In some cases, it was realised that there was a space-time trade-off, whereby a task could be handled either by using a fast algorithm which used quite a lot of working memory or by using a slower algorithm which used very little working memory. The engineering trade-off was then to use the fastest algorithm which would fit in the available memory.

Modern computers are significantly faster than the early computers and have a much larger amount of memory available (Gigabytes instead of Kilobytes). Nevertheless, Donald Knuth emphasised that efficiency is still an important consideration:

"In established engineering disciplines a 12% improvement, easily obtained, is never considered marginal and I believe the same viewpoint should prevail in software engineering"

Benchmarking: measuring performance

For new versions of software or to provide comparisons with competitive systems, benchmarks are sometimes used, which assist with gauging an algorithms relative performance. If a new sort algorithm is produced for example it can be compared with its predecessors to ensure that at least it is efficient as before with known data—taking into consideration any functional improvements. Benchmarks can be used by customers when comparing various products from alternative suppliers to estimate which product will best suit their specific requirements in terms of functionality and performance. For example, in the mainframe world, certain proprietary sort products from independent software companies such as Syncsort compete with products from the major suppliers such as IBM for speed.

Some benchmarks provide opportunities for producing an analysis comparing the relative speed of various compiled and interpreted languages for example and The Computer Language Benchmarks compares the performance of implementations of typical programming problems in several programming languages.

Measures of resource usage

Measures are normally expressed as a function of the size of the input n.
The two most common measures are:

- **Time:** how long does the algorithm take to complete.
- **Space:** how much working memory is needed by the algorithm. This has two aspects: the amount of memory need by the code, and the amount of memory needed for the data on which the code operates.

Examples of efficient algorithms:

- Quicksort, the first known sorting algorithm with a speed of order *O(nlogn)*
- Heapsort, another fast sorting algorithm
- Binary search, for searching an ordered table
- Boyer-Moor string search algorithm, for finding a string within another string

The size and complexity of industrial strength software systems are constantly increasing. This means that the task of managing a large software project is becoming even more challenging, especially in light of high turnover of experienced personnel. Software clustering approaches can help with the task of understanding large, complex software systems by automatically decomposing them into smaller, easier-to-manage subsystems. There are important research directions in the area of software clustering that require further attention in order to develop more effective and efficient clustering methodologies for software engineering. To that end, the state of the art in software clustering research. the clustering methods that have received the most attention from the research community and outline their strengths and weaknesses.

Ethics Concerns Surrounding This Kind of Analytics – Big Data Ethics

As engineers are the ones who confront questions such as 'Who owns all the data that is being analysed? Are there limits to what kinds of inferences that can be made, or what decisions can be made about people based on those inferences?' daily it is essential to talk about the issues of Big Data revolution in the context of software development.

While there's nothing particularly new about the analytics conducted in big data, the scale and ease with which it can all be done today changes the ethical framework of data analysis. Developers today can tap into remarkably varied and far-flung data sources. Just a few years ago, this kind of access would have been hard to imagine. The problem is that our ability to reveal patterns and new knowledge from previously unexamined troves of data is moving faster than our current legal and ethical guidelines can manage. Certain things that were impossible a few years ago can be now done, and the existing ethical and legal maps have been driven off. Failure to preserve the values we care about in our new digital society may lead to the big data capabilities being at risk of abandoning these values, for the sake of innovation and expediency.

If you consider the recent $16 Billion acquisition of WhatsApp by Facebook. WhatsApp's meteoric growth to over 450 million mobile monthly users over the past four years was in part based on a "No Ads" philosophy. It was reported that Snapchat declined an earlier $3 Billion acquisition offer from Facebook. Snapchat's primary value proposition is an ephemeral mobile message that disappears after a few seconds to protect message privacy. Why is Facebook willing to pay Billions for a mobile messaging company? *Demographics and Data.* Instead of spending time on Facebook, international and younger users are increasingly spending time on mobile messaging services that don't carry ads and offer heightened privacy by design. In missing this mobile usage, Facebook is lacking the mobile data. With WhatsApp,

Facebook immediately gains access to the mobile data of hundreds of millions of users and growing. While WhatsApp founder Jan Koum promises “no ads, no games and no gimmicks” and has a board seat to back it up, Facebook has a pretty strong incentive to monetize the WhatsApp mobile data it will now control.

Big Data is about much more than just correlating database tables and creating pattern recognition algorithms. It’s all about money and power. Big Data, broadly defined, is producing increased powers of institutional awareness and power that require the development of what we call Big Data Ethics. The Facebook acquisition of WhatsApp and the whole NSA affair shows just how high the stakes can be. Even when we’re not dealing in national security, the values we build or fail to build into our new digital structures will define us.

From my perspective, I believe that any organizational conversation about big data ethics should relate to four basic principles that can lead to the establishment of big data norms:

- **Privacy isn’t dead*;*** it’s just another word for information rules. Private doesn’t always mean secret. Ensuring privacy of data is a matter of defining and enforcing information rules – not just rules about data collection, but about data use and retention. People should have the ability to manage the flow of their private information across massive, third-party analytical systems.
- **Shared private information can still remain confidential.** It’s not realistic to think of information as either secret or shared, completely public or completely private. For many reasons, some of them quite good, data (and metadata) is shared or generated by design with services we trust (e.g. address books, pictures, GPS, cell tower, and Wi-Fi location tracking of our cell phones). But just because we share and generate information, it doesn’t follow that anything goes, whether we’re talking medical data, financial data, address book data, location data, reading data, or anything else.

- **Big data requires transparency.** Big data is powerful when secondary uses of data sets produce new predictions and inferences. Of course, this leads to data being a business, with people such as data brokers, collecting massive amounts of data about us, often without the knowledge or consent, and shared in ways that are not wanted or expected. For big data to work in ethical terms, the data owners (the people whose data we are handling) need to have a transparent view of how our data is being used – or sold.
- **Big Data can compromise identity.** Privacy protections aren't enough anymore. Big data analytics can compromise identity by allowing institutional surveillance to moderate and even determine who we are before we make up our own minds. Starting to think about the kind of big data predictions and inferences that will be allowed, and the ones that will not.

There's a great deal of work to do in translating these principles into laws and rules that will result in ethical handling of Big Data. And there's certainly more principles that need to be developed as more powerful tech tools are built. But anyone involved in handling big data should have a voice in the ethical discussion about the way Big Data is used. Developers and database administrators are on the front lines of the whole issue. The law is a powerful element of Big Data Ethics, but it is far from able to handle the many use cases and nuanced scenarios that arise. Organizational principles, institutional statements of ethics, self-policing, and other forms of ethical guidance are also needed. Technology itself can help provide an important element of the ethical mix as well. This could take the form of intelligent data use trackers that can tell us how our data is being used and let us make the decision about whether we want our data used in analysis that takes place beyond our spheres of awareness and control. Clear default rules are needed for what kinds of processing of personal data is allowed, and what kinds of decisions based upon this data are acceptable when they affect people's lives. But the important point is this – a big data ethics is needed, and software developers need to be at the centre of these critical ethical discussions. Big data ethics, as I have argued here, are for everyone.

Bibliography

- Florac, W.A., Park, R.E. and Carleton, A.D., 1997. *Practical software measurement: Measuring for process management and improvement* (No. CMU/SEI-97-HB-003). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

- Kan, S.H., 2002. *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc.. Can be accessed at: http://www.informit.com/articles/article.aspx?p=30306&seqNum=5

- Bryant, S.P., Solano, E., Cantor, S., Cooley, P.C. and Wagener, D.K., 2011. Sharing research models: using software engineering practices for facilitation. *Methods report (RTI Press)*, *2011*, p.1. Can be accessed at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3116654/

- https://en.wikipedia.org/wiki/Algorithmic_efficiency#Examples_of_efficient_algorithms

- Shtern, M. and Tzerpos, V., 2012. Clustering methodologies for software engineering. *Advances in Software Engineering*, *2012*, p.1. Can be accessed at: https://www.hindawi.com/archive/2012/792024/

- Knuth, Donald (1974), "Structured Programming with go-to Statements", *Computing Surveys*, ACM, **6** (4)

- King, J.H. and Richards, N.M., 2014. What's up with Big Data ethics. Can be accessed at: http://radar.oreilly.com/2014/03/whats-up-with-big-data-ethics.html