Myths and Over-simplifications in Software Engineering

Magne Jørgensen

***Abstract*— The software engineering discipline contains numerous myths and over-simplifications. Some of them may be harmless, but others may hamper evidence-based practices and contribute to a fashion- and myth-based software engineering discipline. In this article we give examples of software engineering myths and over-simplifications and discuss how they are created and spread. One essential mechanism of the creation and spread of myths and over-simplifications are, we argue, people's tendency towards searching for confirming and neglecting disconfirming evidence. We report from a study examining this tendency. The study demonstrated that the developers who believed in a positive effect of agile methods tended to interpret randomly generated (neutral) project data as evidence confirming the benefit of agile methods. For the purpose of supporting evidence-based practice and avoiding unwanted influence from myths and over-simplifications, we provide a checklist to be used to evaluate the validity of software engineering claims.**

***Index Terms*—Confirmation bias, evidence-based software engineering, myths, over-simplifications.**

I. Introduction

Important decisions and judgments in software engineering should be based on relevant evidence, collected from practice and research studies, *not* on myths, over-simplifications or irrelevant evidence. A necessary condition for the software engineering discipline to become more evidence-based, which is the goal of this paper, is that software professionals and researchers improve their ability to evaluate when claims are valid and relevant and when they are based on myths or over-simplifications [1].

Section II provides a selection of myths and over-simplifications currently present in the software engineering discipline. The main purpose of this section is to motivate the need for more evidence-based practice in software engineering and to illustrate the main mechanisms leading to the creation and spread of myths and over-simplifications. Section III reports results from an experiment that demonstrates the effect of one of the mechanisms leading to a belief in myths and over-simplifications, i.e., the confirmation bias mechanism. Section IV suggests a checklist for the purpose of evaluating the validity of software engineering claims.

Manuscript submitted August 15, 2012.

Magne Jørgensen is with Simula Research Laboratory, Oslo, Norway (e-mail: magnej@simula.no)

II. Examples of Myths and Over-simplifications

Myths and over-simplifications may be defined as incorrect claims. While a myth is plain wrong, an over-simplification may have contexts where it is true or at least contexts where its intended message can be useful. With increased vagueness a myth or over-simplification may turn into a non-falsifiable claim. In this section we focus on presenting and discussing myths and over-simplification, but we will also briefly illustrate and discuss non-falsifiable claims.

A. Most of Our Communication is Non-verbal

It is frequently claimed that most of our communication is non-verbal, i.e., body language and voice. A variant of the claim is the seemingly more precise claim that as much as 93% of our communication is non-verbal, see for example [2, page 53]. A high importance of non-verbal communication may, for example, motivate training in non-verbal communications, extreme awareness of non-verbal cues and to argue in favour of having more expensive, but richer in terms of communication, face-to-face project meetings instead of communicating through emails.

Most people may have experienced situations where the non-verbal communication was essential for the final outcome. Our ability to recall such confirming situations, together with our tendency towards not searching for situations where the verbal communication clearly was the most important may be sufficient to make many accepting the claim [3]. Few people, as far as we have experienced, ask the following critical questions about the claim: What is the precise meaning of the claim? How strong is the evidence in support of it?

A precise meaning of the claim requires, amongst others, that we have the same measure for the *amount* of non-verbal and verbal communication. Otherwise, we would not be able to claim that *most* communication is non-verbal. If do not have this common measure, which in many situations will be hard, the claim makes little meaning. Studies comparing verbal and non-verbal communication, however, typically compare how much each communication source dominates a particular type of outcome in a particular context. Already at this stage, we should start doubting the meaningfulness of the claim in its typical format. If we cannot compare the amount of verbal and non-verbal communication, what is then the meaning of "most communication is non-verbal"?

In many situations with myths and over-simplifications there will be no reference to evidence. In this case, however, there seems to be an agreement about that the main source of evidence, at least for the 93% claim, is a paper from 1967 written by Mehrabian and Wiener [4]. The design of this study was as follows: The participants hear a person saying

the positive words "honey", "thanks" and "dear", the neutral words "maybe", "really", "oh" and the negative words "don't", "brute" and "terrible". All these words are spoken in a positive, neutral and negative voice and the participants are asked to assess "… *the feelings of a speaker towards the person whom she is addressing*". The authors found, not surprisingly, that if a person said, for example, "thanks" in a negative voice to another person, most observers would *not* emphasise the positive content of the word, but instead the negative voice, when assessing to what extent the speaker liked the other person. This study, together with a follow-up study with facial expression added, also published in 1967, are the sources of the claim that 93% of communication is non-verbal. We doubt, however, that anybody who had actually read and understood the design of the original studies would believe that we could use this study as evidence in support of the general claim that 93% (or most of) of communication is non-verbal.

B. 189% Average Cost Overrun

The perhaps most cited report on software cost overruns is the Standish Group's Chaos Report from 1994, which claims that the average cost overrun in software projects was as high as 189% [5], i.e., the actual cost was on average almost three times higher than the estimated cost! Although from 1994, the study is even today used to document a software project crisis, see for example [6]. Interestingly, the tabular data presented in their report is consistent with an average cost overrun of about 89% (or 189% of the original cost estimate) and it seems as if the Standish Group has confused not only the readers, but also themselves, by mixing "189% of original cost overrun" with "189% cost overrun". An 89% cost overrun is, however, also substantial and indicates a software crisis, given that we can trust the result.

The evidence to support the claim of 189% cost overrun is based on a survey of projects from as many as 365 software companies. A high number of respondents is, however, not automatically leading to high reliability of results. An examination of the study design shows that the Standish Group did explicitly ask for *non-representative* projects (page 13 of their report): "*We then called and mailed a number of confidential surveys to a random sample of top IT executives, asking them to share failure stories.*" Clearly, a collection of failure stories cannot be used as a representative sample of the software industry's projects. The 189% cost overrun result, consequently, only tell something about the average cost overrun of the projects with large cost overruns, which is far from how it has been and still is presented. More about the problems with the Standish Group's study design in [7].

It may be useful to reflect on why so many software engineering researchers and professionals accepted and spread the Standish Group-created myth of extremely high average cost overrun in software projects. Researchers and software professionals could have referred to one of the many other surveys on cost overruns, e.g., [8, 9], instead of the results of the Standish Group report. The other surveys of software cost overruns show, quite consistently, average cost overruns in the interval 20-40%. A likely reason for the selection of the much higher cost overrun values of the Standish Group is that the extreme data were more useful for the purpose of selling advisory work and demonstrating the importance of more research on cost estimation. The willingness and ability to search for disconfirming evidence and be critical towards a claim may decrease with increased benefit from the claim being true.

C. Brook's Law

Brook's law says that: "*Adding manpower to a late software project makes it later*" [10]. Brooks himself describe his own statement as an outrageous over-simplification, but this has not stopped other people from interpreting "Brook's law" as an actual law, or at least a rule with very few exceptions. It is, for example, used as support for the a context-independent trade off-function between time and effort in cost estimation tools [11]. While it may be true that adding inexperienced people to a late project usually makes it later, there are, as far as we know, no evidence documenting that adding skilled, experienced people to a late project in general makes the project later. Adding people to a late project is what the software industry does, as far as we have experienced, and what seems to be a fully rational behaviour in many contexts.

What makes the use of Brook's law an interesting example of over-simplification is, we think, that it can be used positively to remind project managers and leaders that adding manpower in software development does not automatically lead to reduced time to completion of the project. People new to the project may have to be trained and they are likely to lead to an increase proportion of effort spent on project member communication. The problem occurs when we do not acknowledge that Brook's law is an outrageous over-simplification, and use it to argue against rational behaviour to rescue challenged projects. Of the mechanism leading to the spread of over-simplifications, such as Brook's law, may be our wish for simple rules, i.e., rules that require us to reflect less on the context and do less hard thinking.

D. Factor Ten Increase in Error Correction Cost

Some people claim that there is an increase of a factor of about ten to correct error effort per phase, see for example the claims by the XP-expert Alistair Cockburn *xprogramming.com/articles/cost_of_change*. An error that costs one hour to correct in the requirement phase would according to this claim typically take about 10 hours to correct in the design, 100 hours in the programming and 1000 hours in the test phase. There are less extreme claims, e.g., the claim that the correction cost is higher (perhaps with a factor less than ten) the later the error is detected [12]. The claim of increasing cost is frequently used to argue that it is always beneficial to detect errors as early as possible, see for example *hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/209/error.htm*.

While we were unable to find any empirical evidence documenting the factor ten increase in correction cost, there are several studies that find that it takes on average more effort to correct errors detected in later phases, see for example [13]. So what is the problem with the claim?

The claim is, we argue, an over-simplification of the research results. The studies, for practical reasons, observe

the average cost of error correction in the phase where the error was actually detected. The studies do, however, not say anything about how much these errors would have cost to identify and remove in an earlier phase. Removal of the last requirement and design errors, for example, is likely to cost very much more than removal of the average requirement and design errors. In agile development the insight of increased marginal cost of detecting and removing requirement errors in the requirement phase has led to an approach where it is planned with frequent correction of the requirement specification, instead of trying to remove all of them before starting the design phase. If it always were beneficial to remove errors in the phase where they are introduced, especially given a factor ten increase in cost per phase, agile development would probably be very inefficient.

An example of a misuse of this over-simplification can be found in the perhaps most quoted report on the benefit of better testing infrastructure, i.e., the NIST report on "*The economic impacts of inadequate infrastructure for software testing*" [14]. This study calculates a very high benefit from improved testing infrastructure (22.2 – 59.5 billion USD) and is, not surprisingly, used by many researchers and advisors to strengthen the need for research funding on testing and motivate the need for advisory services. The report states (page 5-4) "*... regardless of when an error is introduced it is always more costly to fix it downstream in the development process.*" An assumption of no added error detection cost and always decreasing correction cost when detecting more errors in the phases where they are introduced, creates, not surprisingly, great savings from better testing infrastructure. The assumption is, as argued earlier, not supported with evidence, and, likely to be incorrect.

The claim of increasing cost of correcting errors in later phases has, similarly to Brook's law, a core of essential truth. A brief question to the client to clarify a requirement may cost a few minutes of work, while the discovery of a requirement error related to misinterpreting a requirement may cost many work-hours to correct in the testing phase. The over-simplification does however potentially also make harm since it removes the focus from the more precise and more essential questions: How much and what kind of early error detection and removal is cost-efficient for a particular context (type of system, development method, complexity of development, stability of requirements)? As with Brook's law we have the desire for simple rules. Unfortunately this may frequently not be consistent with a complex software engineering world full of context-dependent insights.

E. *45% of Features Never Used*

The Standish Group claimed, in a presentation at the conference XP 2002, that 45% of the features of software applications were never used, when developed by "traditional" (waterfall) development teams (no official report exists, as far as we know). The claim has been used extensively to argue against the use of "traditional" development and to support a transition towards more use of agile development teams, see for example thecriticalpath.info/2011/07/07/waste-in-software-projects/. Agile development methods embrace frequent update of the requirement specification based on feedback and learning during the project to avoid delivery of useless, originally specified features.

There are several reasons not to trust the finding by the Standish Group:

i) The study design is not publicly available. We were for example unable to find out how and who they asked about the information. In order to trust the results, the respondents of the survey by the Standish Group should be well informed about previous, current and future use of the application for *all* users. Neither the developers nor single users are typically that well informed.

ii) The claim is difficult to interpret. Those who use the claim to promote agile methods typically use it to mean that 45% of the features has, for the average system, never been used by anybody. This would be a chocking result suggesting that something is very wrong with how we develop software, i.e., a software crisis motivating a change to new development methods, while the claim that each individual user on average use only 45% of the functionality would perhaps not be surprising or bad at all. The spread of the undocumented, probably incorrect claim seems to be supported by a benefit from it being true among agile method supporters.

F. *Productivity Ratio of 1:10*

It is sometimes claimed that the productivity ratio between the least and the most productive programmer is about 1:10 (or some other ratio). The factor 1:10 may have its origin in a study from 1968 [15], which states that: "*These studies revealed large individual differences between high and low performers, often by an order of magnitude.*"

It may be debated to what extent the claim of factor ten is a myth, an over-simplification or simply a meaningless statement. It is easy to see that you can get any productivity difference number out of empirical studies comparing programmers. The ratio between the least and the most productive programmer is for example likely to increase with higher number of programmers and with more complex tasks. If a task is complex enough, the least skilled developers may not be able to complete it (undefined productivity) or complete it incorrectly so that other programmers has to correct it (possibly negative productivity). For a good discussion of this claim, see *forums.construx.com/blogs/stevemcc/archive/2008/03/27/productivity-variations-among-software-developers-and-teams-the-origin-of-quot-10x-quot.aspx*

As with several of the other claims, the intention of this claim is likely to be good. On the other hand, to believe in a fixed, task-independent, proportion between the best and the worst may be counter-productive and, for example, lead to under-estimation of the performance difference on really complex tasks. Our quest for simple rules may be understandable, but also in this case the context matters and productivity ratios without knowing how many developers who were included in the study and the complexity of the task is not very informative.

G. Non-falsifiable Claims

Most software engineering claims with high impact on practice may neither be myths or over-simplifications, but rather belong to the category of non-falsifiable claims. Non-falsifiable claims have many similarities with myths and over-simplifications, especially when it comes to the type of control questions useful to evaluate their validity.

One example of a non-falsifiable claim, possibly with high impact on practice (formulated as a value in the agile development manifesto) is the following: "*... we have come to value ... working software over comprehensive documentation.*" This value, which implicitly is a claim about what is a more efficient work process, may have contributed to less effort spent on documentation in agile than in most other types of software projects. The problem with the claim is that it falls into one out of the following two categories: i) The category of obviously correct claims, i.e., who would not rather have software that works (without comprehensive documentation) than comprehensive documentation (without software that works)? ii) The category of non-falsifiable claims, i.e., if the term "comprehensive documentation" is interpreted as inefficient or too extensive documentation, it will not be possible to falsify the validity of the claim. If a project benefit from some documentation, it could easily be claimed that this documentation is not what is meant by "comprehensive documentation" and do not falsify it.

H. Mechanisms

Summarized, we find that central mechanisms for the creation and spread of myths and over-simplifications include:

- Confirmation bias, i.e., we see patterns that are not there if we expect or want to see them ("we see it, when we believe it")
- Poor studies, e.g., use of non-representative samples
- Misunderstood or over-generalized research results
- Usefulness bias, i.e., we benefit from a claim being true and are for this reason less motivated check its validity properly
- Insufficient check of the validity, scope and robustness of the evidence, e.g., not reading the original study leading to the claim
- Poor precision level of claims, which makes it easier to recall confirming evidence and more difficult to falsify
- A tendency towards interpreting a claim with the intention to believe rather than disbelieve it
- Desire for simple, deterministic relationship
- Belief in authorities
- Repetitions, i.e., the more frequently a claim is repeated, the more we believe in it, even when all claims are based on the same, perhaps misunderstood, source of evidence.

III. A Confirmation Bias Experiment

To illustrate the effect of confirmation bias, i.e., the effect of "we see it, when we believe it", we conducted an experiment with software professionals. The participants consisted of 50 software developers employed in a company that had recently started using agile methods in many of their software development projects.

A. Design

The experiment consisted of the following three steps:

i) Collection of the developers' beliefs in the effect of agile methods compared to "traditional" (waterfall-based) methods.

ii) Generation of ten different data sets. The data were generated *randomly*, i.e., there was no systematic difference in performance, measured as productivity and user satisfaction, between the projects using agile and traditional development methods. Each data set contained outcome data of twenty projects, ten of them using agile and ten of them traditional development methods.

iii) Provision of one, randomly allocated, data set to each of the developers with the request: *"Assume that this [the data set] is the only information you have about the use of agile and traditional development methods in the company and that you are asked to interpret the data. The organization would like to know what the data shows related to whether they have benefited from use of agile methods or not."* We emphasized in the instructions that the developers were supposed to interpret the data alone, i.e., "what the data shows", and not what they believed would be the case in general or combine it with their own experience. We told the participants that the data was from another, i.e., not their own, company. The developers' responses should be based on the selection of a number between 1 (agree strongly) and 5 (disagree strongly), reflecting how much they agreed with the statement "*I believe that the data set indicates that the company has benefited from the use of agile methods.*" A neutral answer ("no difference"/"don't know") would give the response 3.

B. Results

The measurement of the developers' belief about the benefits of agile methods, which was conducted before they were exposed to the data sets, gave that 84% of them believed that agile development methods in general had a positive effect on productivity and 66% believed in a positive effect on user satisfaction. There were a few developers without a strong opinion, i.e., they used the "don't know" option on the effect with respect to effect on productivity (10%) or on user satisfaction (26%). Even fewer developers believed in a *negative* effect of agile methods on either productivity (6%) or user satisfaction (8%). None of the developers believed in a negative effect on both productivity and user satisfaction. In total this shows a strong tendency towards believing in the benefits of agile methods among the developers. We categorized the 18 developers who stated that they had no opinion (don't know) or believed that there was a negative effect on either productivity or user satisfaction, as "undecided". The other developers were categorized as "believe in agile".

A random generation of project data with only twenty data points does not guarantee that there are no patters in favour of either agile or traditional development in some of the data sets. An examination of the randomly generated data set gave that the five of the data sets were close to neutral on both productivity and user satisfaction. One data set was in favour of agile development and one data set was in favour of traditional development for both productivity and user satisfaction. The remaining three data sets were

mixed, i.e., in favour of one development approach for productivity and in favour of the other approach for user satisfaction. The average values of productivity and user satisfaction when joining all ten data sets were not significantly different for the two development methods.

If the software developers', in general, positive attitude towards agile methods had no impact on the responses, we would expect an average response of about 3 ("no difference"/"don't know"). What we found, however, was a median response of 2 ("agree"). The effect of the previous belief on data interpretation, i.e., the confirmation bias effect, is further strengthened when comparing the responses of the developers with positive view on agile methods with those of the undecided. A chi-square analysis of the variable PreviousBelief ("undecided" vs "positive to agile") and DataInterpretation ("agree strongly" + "agree" vs "disagree" + "disagree strongly") gives that independence between previous belief and data interpretation is highly unlikely (p=0.1). Among the developers who were positive to agile development, 60% interpreted the data set to show a positive effect, 22% a negative effect and 20% found no effect of agile methods. Among the developers who were undecided towards the benefits of agile development, 40% interpreted the data set to show a positive effect, 40% a negative effect and 20% found no effect. In short, those undecided about the benefits of agile development were, on average, unbiased in their data interpretation, while those with a previous positive attitude towards agile methods interpreted the data sets to show what they might have expected to see in the data sets, i.e., a positive effect of agile methods.

In real life contexts, there will typically be more than two relevant variables and more subjectivity in the interpretation than in our experiment. This increase in interpretation complexity and subjectivity is, we believe, likely to increase the strength of a confirmation bias in real life contexts. It is therefore, we believe, remarkable that even in our simple experimental situation, with interpretation of only two, objectively measured variables, there are more than enough opportunities to get affected by previous belief.

IV. How to avoid being affected by myths and over-simplifications

Many myths and over-simplifications may, as partly illustrated by our discussion in Section II, be disclosed by using the following set of control questions:

1. What does the claim means? When is it supposed to be valid? Has it any precise meaning? Is it possible to falsify it?
2. Before looking at the evidence related to a claim, ask yourself what you would expect of supporting evidence to believe in it. Is this evidence, or equally convincing evidence, present?
3. Do you have a previously formed opinion about the validity of the claim? If so, be especially aware of your tendency to accept poor evidence and argumentation.
4. Who makes the claim? Do you accept it because of the authority claiming proposing it, or based on the validity of the supporting evidence?
5. What is the quality of the evidence? Is the evidence neutral, as opposed to collected by people with vested interests in a particular outcome, and relevant? Is there any evidence at all? Search for confirming and disconfirming evidence. The evidence should be as neutral as possible. If possible identify and understand the original source, e.g., study or practice-based experience, of the evidence.
6. What does the totality of evidence say? How strong is it and in which contexts is it valid?

V. References:

1. Dybå, T., B. Kitchenham, and M. Jørgensen, *Evidence-based software engineering for practitioners.* IEEE Software, 2005(Jan-Feb): p. 58-65.
2. Borg, J., *Persuasion: The art of influencing people*2008: Financial Times Press.
3. Nickerson, R.S., *Confirmation bias: A ubiquitous phenomenon in many guises.* Review of General Psychology, 1998. **2**(2): p. 175-220.
4. Mehrabian, A. and W. Morton, *Decoding of inconsistent communication.* Journal of Personality and Social Psychology, 1967. **6**(1): p. 109-114.
5. *The Standish Group: Chaos Chronicles Version 3.0*, 2003, The Standish Group: West Yarmouth, MA.
6. Doloi, H.K., *Understanding stakeholders' perspective of cost estimation in project management.* International Journal of Project Management, 2011. **29**(5): p. 622-636.
7. Jørgensen, M. and K. Moløkken-Østvold, *How large are software cost overruns? A review of the 1994 CHAOS report.* Information and Software Technology, 2006. **48**(4): p. 297-301.
8. Yang, D., et al. *A Survey on Software Cost Estimation in the Chinese Software Industry*. in *ACM/IEEE International Sympsoium on Empirical Software Engineering and Measurement*. 2008. Kaiserslautern, GERMANY: Assoc Computing Machinery.
9. Moløkken-Østvold, K., et al. *A survey on software estimation in the Norwegian industry*. in *10th International Symposium on Software Metrics*. 2004. Chicago, IL: IEEE Computer Society.
10. Brooks, F.P., *The mythical man-month: Essays on Software Engineering*1975, Reading, Mass.: Addison-Wesley. XI, 195 s. : ill.
11. Nan, N. and D.E. Harter, *Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort.* IEEE Transactions on Software Engineering, 2009. **35**(5): p. 624-637.
12. Westland, J.C., *The cost behavior of software.* Decision Support Systems, 2004. **37**: p. 229-238.
13. Boehm, B.W., *Verifying and validating software requirements and design specifications.* IEEE Software, 1984. **1**(1): p. 75-88.
14. *The Economic Impacts of Inadequate Infrastructure for Software Testing*, 2002, National Institute of Standards and Technology.
15. Sackman, H., W.J. Erikson, and E.E. Grant, *Exploratory Experimental Studies Comparing Online and Offline Programming Performance.* Communications of the ACM, 1968. **11**(1): p. 3-11.