ABSTRACT
In efforts to reduce the number of faults in software, there has been much focus on technical approaches, such as fault prediction schemes. These approaches have now reached a ceiling in their efficacy and 20% of faults cannot be recalled even with the best prediction techniques currently available. In the end, faults in software are caused by human software developers; in most cases, this is either because requirements have been misunderstood or because of human error in the implementation, i.e. a planned sequence of activities fails to achieve its intended outcome. Our interest is in trying to identify what kinds of faults are generated by what kinds of individuals. We believe that there is potential to leverage human error theory and research into human characteristics of software developers in achieve this and hence be able to reduce the software faults introduced into code.

Categories and Subject Descriptors

General Terms
Management, Performance, Human Factors.

Keywords
Software faults, human characteristics.

1. INTRODUCTION
Software faults continue to have a significant impact on people and on the success of businesses. The serious consequences of software faults are regularly reported in press headlines, a prime example of which includes the recent failure of NATS (the UK’s National Air Traffic Services).\(^1\) The cost of finding and fixing faults is high. Operational faults in commercial products have been reported to cost $7k per fault to fix [24] with software developers spending nearly half their time fixing faults [21].

Current approaches to reducing software faults are technocentric and dominated by testing techniques, fault prediction modelling and inspection approaches, each with limited success. For example, prediction techniques recall a maximum 80% of faults; 20% of faults cannot be recalled even with the best prediction techniques currently available [16]. Current approaches are limited by two problems. First, they retrospectively find faults already engineered into the software. Second, no account is taken of the human originating the faults.

Our suggestion is to identify what kinds of faults are generated by what kinds of developers, so that appropriate mechanisms (tools, management, profiling systems) may be developed to help avoid faults being introduced rather than retrospectively finding faults. This is a human rather than technical approach. Software faults\(^2\) are caused by developer errors\(^3\); making their origin human rather than technical. The importance of human issues in software engineering has been recognised for decades (e.g. [10]), but it is only in recent years that research into this area has received significant attention (e.g. [1], [15]). Despite this increase in general, few studies report on how developers’ characteristics affect the software faults they generate. Those studies that do, offer only a high level analysis of a very limited subset of the factors involved, e.g. [19], [26]. Results from the most recent study in the area [19] indicate that such an approach can yield positive results, but more can be done. Previous studies recognise the need for a more comprehensive understanding of the relationship between developers’ human characteristics, the errors they make and the diversity of faults introduced into code.

Human mistakes and the role of human error theory have been widely investigated in other disciplines (e.g. [30]) where controlling for mistakes is considered fundamental [31]. The importance of understanding mistakes is an emerging important topic in software engineering.

It is now time to take the existing exploratory work a step forward: to extend and deepen current understanding of the relationship between developer human characteristics and faults.

2. FAULT CHARACTERISTICS
The characteristics of faults have been studied for over 30 years during which time a large collection of fault characteristics schemes have been published. Pliski et al [27] provide an overview of these schemes. Some schemes are high level, offering a general view of faults (e.g. [8]) others are detailed and contain hundreds of fault classifications (e.g. [3]). Some schemes are application area specific (e.g. Palix et al’s [25] Linux scheme), while others are lifecycle phase specific (e.g. NRC’s requirements fault taxonomy or Beizer’s [3] testing scheme).

Previous studies investigating relationships between faults and features of code largely treat faults as homogenous and

---

\(^1\)http://www.caa.co.uk/docs/2942/s%3A%20%20Intermediate%20Report%20-%20NATS%20System%20Failure%2012%20December%202014.pdf

\(^2\)Faults can occur in any software artefact but for practical reasons we limit our scope to the study of code faults.

\(^3\)We use the standard IEEE definition of fault as an imperfection in the software: a failure as the manifestation of the fault in operational use; an error as the human behaviour leading to a fault; mistakes, slips and lapses being particular elements of errors [30].
concern themselves only with fault numbers (e.g. [23]). However more recent work recognises that faults are not homogeneous and have many characteristics (e.g. [29]). It seems likely that different human characteristics underpin different types of fault. Our ability now to identify the characteristics of faults offers new insights into faults and provides a potential basis for relating them to the human characteristics underpinning them.

3. DEVELOPER CHARACTERISTICS

Although the relationship between individual human characteristics and faults has not been investigated in detail, previous research suggests the following list of characteristics that may be relevant to fault generation: programming skill, programming knowledge, general mental ability, cognitive style, personality, motivation and cognitive bias.

Programming skill has a significant effect on programmer performance [4]. Bergersen et al. [4] produced an instrument to directly measure programmer skill that we will use in this study. Knowledge and experience play a key role in developer performance (e.g. [34], [33]). Developer programming knowledge may be assessed through specifically-designed tests.

General mental ability (GMA). The higher an individual’s GMA, the stronger the relationship between GMA and job performance [14]. The GAT (General Ability Test) may be used to assess GMA.


Personality affects developer behaviour [17] and individual performance [9]. Several personality tests exist, such as the Big Five model [2].

Motivation has been found to affect software quality [6], and hence faults. Retrospective interviews focused on specific situations [32] have been used to investigate the developers’ motivational circumstances at the time of fault generation.

Cognitive bias leads to systematic and predictable errors [36]. In software development a range of cognitive biases have been identified (e.g. [20], [35]). Confirmation bias has been found to have an impact on software quality, and a methodology has been developed [7] for measuring confirmation bias.

4. A WAY FORWARD

We suggest that a combination of human error theory, developer human characteristics and fault characteristics will allow us to understand and then model the relationship between ‘types’ of developer and the ‘types’ of faults. Several challenges and opportunities arise in this approach.

4.1 Challenges

We see (at least) three challenges with this approach:

Challenge One: There are many human issues that impact on software development, and it is crucial that we focus on the most significant factors likely to contribute to faults. However, it is currently not known what impacts specifically on developers’ propensity to introduce faults in code. The initial set of potential developer characteristics described above arises from the literature, but it will be important to find those which have the most impact.

Challenge Two: There are many fault classification schemes but none of them have been developed with human characteristics in mind. It is not clear which, if any, of the existing schemes will be suitable for this purpose. This challenge gives us the opportunity to build on existing work and to generate a scheme with relevant human characteristics at its core.

Challenge Three: Relating the human characteristics to fault characteristics is a multi-faceted problem and one that will require sophisticated analysis and modelling techniques. One possibility is to use Bayesian Networking (BN) analysis [13], but this has not been used for this kind of analysis and modeling before.

4.2 Opportunities

We see (at least) two opportunities with this approach

Opportunity one: Human error theory distinguishes between ‘slips and lapses’ and ‘mistakes’ [30]. Slips and lapses are errors resulting from a failure of execution ((lapses are usually memory based errors and slips are not-as-planned actions, eg a slip of the pen)). Mistakes are errors resulting from judgmental or inferential processes in the selection of an objective or the specification of the means to select it. A crude but effective way of classifying faults in code is to think of them as being one of two types: those that result from errors being made in the code implementation; and those that result from a misunderstanding of requirements or design specifications. The former may be viewed as slips or lapses and the latter as mistakes.

Opportunity two: Instruments to measure the human characteristics listed in section 3 already exist and could be applied relatively straightforwardly. One example would be to use the Group Embedded Figures Test (GEFT) measure of field independence. This human characteristic is known to indicate whether a person’s cognitive style is to take account of the context of the problem rather than to focus exclusively on the problem itself. It would be interesting to test the hypothesis that someone with high field independence (i.e. they do not take context into account) may make more mistakes and someone with low field independence may make more slips.

5. REFERENCES


[33] Sharp H. 1991, The role of domain knowledge in software design, Behaviour & Information Technology, Vol. 10, Iss. 5


