Towards GEEZMO: hiGh-frEquEncy Zest and Mood-pOlling for Proactive Software Development Problem-Solving

Elisabetta Di Nitto, Raffaela Mirandola, Santi Raffa, and Damian A. Tamburri
Politecnico di Milano
Via Golgi 42
Milan, Italy
{elisabetta.dinitto, raffaela.mirandola, damianandrew.tamburri}@polimi.it, santi.raffa@mail.polimi.it

ABSTRACT

Development of software is happening on an increasingly distributed fashion. Individuals normally coordinate and interact over any combination of IRC rooms, mailing lists, private email, etc. Conversely, big players like Google Inc. employ yearly Googlegeist polls asking its employees how they feel about the company, its directions and its managers. As a consequence, the amount of time required for managers and management to feel the pulse of their subordinates is removed from actual work. We argue that software development status can be computed by eliciting just a few bits of information that can be anonymously extracted by a single poll - the poll can be completed very quickly and its administration can happen as a simple prompt inside windows that developers would be visiting anyway (e.g., the log-in screen, commit screens, etc.). This paper elaborates this idea and develops a prototype to articulate the idea in practice. Finally, the idea is discussed using a preliminary validation by means of statistical methods and simulations.

Categories and Subject Descriptors

K.6.1 [**Project and People Management**]:

General Terms

Theory

Keywords

Mood-Sensing, Social Software Engineering

1. INTRODUCTION

Development of software is happening on an increasingly distributed fashion [10]. On one hand, open-source software development happens almost invariably on multiple locations, involving individuals who do not know each other other than by their screen names and email addresses. Individuals coordinate and interact over any combination of IRC rooms, mailing lists, private email, bug trackers,etc.

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804383

On the other hand, closed source software development is following this trend, e.g., by means of Global Software Engineering (GSE) [22]. As a notable example, Stack Exchange Inc. is the company behind a popular, closed source Q&A system. Their developers are distributed across multiple continents [5]; the company relies on a proprietary chat system, Google Hangouts and the online bug-tracking system Fogbugz for coordination. While some of the core components are made open-source [6], most of the development happens behind closed doors.

As a result, communication between team members happens at the same time over multiple communication channels of several different types and goals, ranging from written and formal communication (mailing lists, bug trackers) [23].

The main challenge here is that most of those systems cannot be adequately monitored or data-mined, either for technical or administrative reasons. A manager wishing to proactively diffuse situations before they detonate would have a difficult time being in the right place at the right time. They may read IRC logs, or parse them using so-called opinion mining [18], but it is unfeasible to do so on every system the team may use for lack of adequate APIs and the possibility of private communication being legitimately unavailable to a manager due to expectations of privacy.

As a result, managers usually rely on regular one-on-one meetings with their developers, e.g., in order to be aware of any organisational and socio-technical issues [21] as they may come up. Additionally, companies invest significant resources in infrequent employee satisfaction surveys in order to read their workforce morale; for example, Google Inc. employs yearly Googlegeist polls [2] asking its employees how they feel about the company, its directions and its managers.

Any amount of time required for managers and management to feel the pulse of their subordinates is removed from actual work. As a result, there is a very clear trade-off to be made between polling "bandwidth" and "frequency": running one-on-one meetings on a daily basis would be a questionable use of company time, while administering long opinion polls too frequently might result in understandably decreased response rates and increasing poll satisficing [14].

We find that this trade-off between time resolution and data resolution (curiously reminiscent of the Heisenberg uncertainty principle [9]) is usually resolved in favour of data resolution; however, another physical principle that may apply here is the Nyquist rate [11]: employee sentiment can vary very dramatically very quickly, and situations may need to be handled within hours of their insurgence. This can become quite complicated when managers have to attend off-

site meetings at times, as they are usually requested. Conflict resolution (if any) is then delegated to the manager of the manager, who may not be in touch with a team's interpersonal relationships; in order to enable them to do so (in addition with manage their managers), they usually have to organise one-on-one meetings of their own.

What if we tried a different trade-off?

This paper proposes GEEZMO, i.e., hiGh-frEquEncy Zest and Mood-pOlling for software development success. By elaborating on GEEZMO, this paper tries to explore the opposite end of the previously mentioned time-data trade-off spectrum: what if project status updates are obtained by asking very little information very frequently? In particular, we propose the use of a daily one-question poll: "How was your day?" The response can be chosen between only a few values (for example, five values ranging from "amazing" to "terrible"). These bits and pieces of information intuitively amount to the "zest" of software developers, i.e., their engagement and energy towards their development scenario and problems as well as their "project mood", i.e., their current status of organisational and social tranquillity they might perceive in that scenario.

On one hand, only a few bits of information can be anonymously extracted by a single poll. On the other hand, this means the poll can be completed very quickly and its administration can happen as a simple prompt inside compatible high-traffic online services you would be visiting anyway. Additionally, since we would not be submitting just one poll, but one poll every day, we gain a lot of implicit information by contextualising the responses with their time frame: spikes in mood might be expected right after a company-sponsored off-site social event, for example; crunching, on the other hand, naturally lowers the developers' quality of life and thus their morale. If you only poll every 12 months, you only can see the aggregate result of one year's worth of events.

This paper illustrates our first attempt at proposing a solution to follow this research path. Finally, we offer a preliminary validation using statistical methods and simulations of a prototype.

2. RESEARCH PROBLEM AND PROPOSED SOLUTION

The research problem we want to tackle concerns how to come up with reasonable estimation of project status using reported many-eyes mood and zest across a software project. Our research question can be phrased as follows: "what project status updates can be obtained by asking very little information very frequently?". The layout of our simple, non-invasive research solution is shown in Fig. 1.

In the following we elaborate the fundamental assumptions and design principles behind GEEZMO, i.e., the solution we propose in pursuit of said research question.

GEEZMO distinguishes between three roles, each corresponding to successive levels in a corporate hierarchy:

- the "voters", people the morale of which we want to monitor;
- the "predictor", a person who is in charge of keeping the morale high;
- the "supervisor", the predictor's manager.

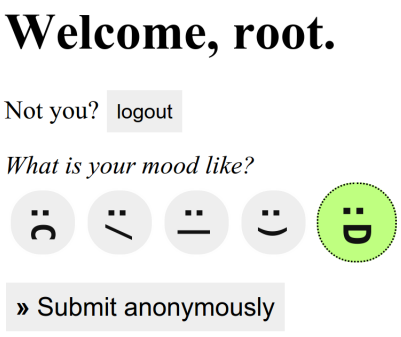

Figure 1: GEEZMO, a Splash-Screen Visual.

GEEZMO asks the voters about their morale, and we shall call that response "vote" by extension. At the same time, it asks the predictor to make an educated guess of his voters' aggregate response; we will similarly dub the predictor's response "prediction".

Distinguishing between votes and predictions allows elaborating possible scenarios:

1. If the two match and they are high, everything is fine.
2. If the two match but they are low, the manager is likely to know what's the issue and we can assume he's working on fixing the issue. There is no cause for further alarm.
3. If the two differ, however, we have a situation that is not being dealt with: the manager's perception of reality does not match what's actually going on. If the situation persists, we warn the supervisor.

Based on these scenarios we are thus looking for a solution that must:

1. Measure the mood of a (relatively) time-independent small population on a short but regular interval;
2. Measure the teams' own awareness of that mood;
3. Raise an alarm when the two diverge;
4. Do so in a time-effective manner, so to minimize the efficiency cost of such a system;
5. Do so in a way that is not self-defeating, that is, the use of GEEZMO shall not accidentally or indirectly punish users for their honesty or pressure them to vote in a given way;
6. Do so in a way that does not cause unnecessary drama, especially when false alarms occur.

In matching the above goals, we elaborated the following design principles:

No pressure. GEEZMO shall not introduce pressure on people to vote a particular way. Particular case shall be added in managing "feedback loops" that result in pressure from outside. For example, by granting anonymity to workers, we seek to reduce consequence for speaking your heart out when filling in the form.

No distraction. GEEZMO shall strive to minimize the cost of doing a mental context switch from actually getting things done to feeding data into GEEZMO.

No miracles. A piece of software that asks a single question and gets a single response can actually do much, but no more than that. Human intervention is still necessary to fix the kind of problems GEEZMO detects and we, indeed, trust a worker's supervisor to do the right thing.

No frills. GEEZMO shall be made available in the easiest form possible, e.g., though the machine log-in or IDE access screen. Additional features within GEEZMO would be unnecessary cognitive burdens.

No trust. Users shall not have to blindly trust GEEZMO or their managers in order to have the peace of mind to vote normally. Data that shall not be disclosed shall not be stored to begin with. GEEZMO shall be resilient against (or at least allow detection of) attempts to sabotage through artificial voting patterns.

No forcing. GEEZMO shall make no assumptions the organisational social structure in the organisation in question, e.g., if it is hierarchical (typical in closed-source) or circular (typical in open-source). In so doing GEEZMO, must be designed both high-level organisational scenarios, namely, hierarchical and circular. Although we tested a preliminary application of this principle in our current prototype, our results suggest it may need further research. More detailed discussion can be found in Section 5.1.

In applying the principles above, we developed an open-source, proof-of-concept implementation written in Django (version 1.6).

The implementation reuses Django built-in standard components such as users, authentication mechanisms, signals, models and templates in an attempt to be as modular as possible. This should ease future deployment into arbitrary environments; for example, replacing Django's built-in authentication back-end with another one requires you to only write a couple of methods, whereas plugging in a different template system is only a matter of importing its Python 2.x bindings from `views.py` and replacing the method calls as needed.

The git repository for this implementation of our proof-of-concept prototype can be found online[1].

3. RAISING ALARMS

We now have devised two different schemes for determining when a prediction is "successful", but that only gets us so far. We also need to decide when enough non-successful predictions become a probable cause for alarm.

If we use 2 as the number of unsuccessful predictions triggering an alarm for hierarchical teams and 5 for circular teams (where all voters can also be predictors), we can consider each day's predictions as respectively a Bernoulli trial with $p = 0.5$ (as calculated in table 1) and a Bernoulli series with $p = 0.4$.

[1] `http://github.com/badp/weather`

Let us consider the number of successful votes s out of the total number of predictions m: we expect $s \simeq pm$. Intuitively, if $s \ll pm$, then there is unexpected unrest, whereas if $pm \ll s \simeq m$, chances are the system is failing to acquire meaningful data from the polling. The Neyman-Pearson lemma [16] confirms that, if we want to test that the maximum likelihood success rate for a binomial series is $\theta = p$ against $\theta \neq p$, the most powerful likelihood-ratio test of size α succeeds if $k_1 < s < k_2$ [4]; in order to derive k_1 and k_2 we can calculate this test's p-value.

In other words, if we are willing to tolerate at most a probability α that the system incorrectly raises an alarm, then the check that can possibly raise the least amount of false alarms succeeds if:

$$P(S = s) = \binom{s}{m} p^s (1-p)^{m-s} \leq \alpha$$

where $S \sim \mathbf{Bi}(m, p)$ is the expected probability distribution of s.

If we consider $t = 5$ to be the number of days the system should take before raising an alarm and $\alpha = 0.05$, we have:

- In the hierarchical case, $m = 5$, $p = \frac{1}{2} = 1 - p$, it follows that $P(S = s) = \binom{s}{5}\frac{1}{32} = \begin{cases} 1/32 & s \in \{0, 5\} \\ 5/32 & s \in \{1, 4\} \\ 10/32 & s \in \{2, 3\} \end{cases}$.
  Thus, we raise an alarm if the number of successes is either 0 or 5.

- In the circular case, we have to apply the formula directly.

  For example, consider a team of $n = 6$ people with $m = 24$ predictions in the last 5 days. Since we are using "five options," $p = 0.4$ and $S \sim \mathbf{Bi}(24, 4)$. By calculating $P(S = s)$ for $s \in \{0, \ldots, 24\}$ we realize that a warning should be raised in case $s \leq 5 \vee s \geq 14$ ($E[S] = mp = 9.6$). Of course, in practice, it is not necessary to calculate all 16 possible values of s: it suffices to compare $P(S = s)$.

Again, in the calculation of these formulas we have made the inaccurate assumption that each prediction's success is independent from other predictions; this is especially a problem in the circular scenario. Future work should seek to take this into account, for example by adopting a suitable approximation of the generalized linear model [15].

As you might notice, GEEZMO has not one but *two* Tables of votes: `Vote` and `TodayVote`. We assume that the `Vote` Table is public, whereas the `TodayVote` Table is private and, in real-life deployments, access to the Table is limited to GEEZMO and only GEEZMO. Votes are added to `TodayVote` and, when midnight rolls, a task processes the contents of the Table. For example, if a team only receives two or less votes in a day, we discard those votes. The order of those votes is also shuffled. The main advantage this measure brings is the guarantee that the current day's votes will not be accidentally leaked by any user-facing views (such as the supervisor's) reading off the `Votes` Table. This measure also allows us to avoid adding a `has_voted_today` flag to the `UserProfile` Table.

The second decision is using two Tables for users: `User` and `UserProfile`. This is mainly a best-practice workaround

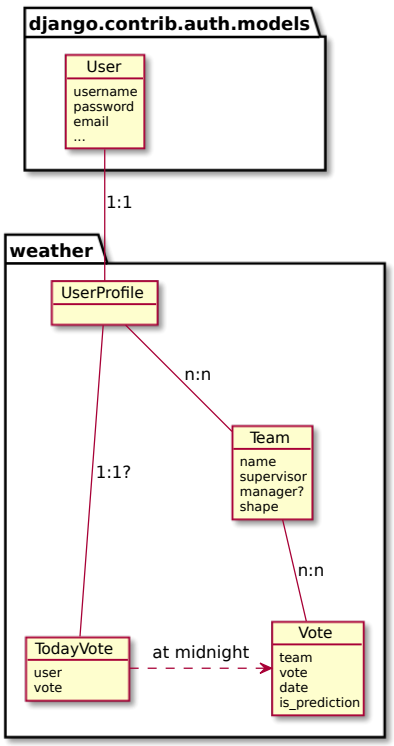

Figure 2: Entity-relation graph for GEEZMO.

on the fact that the `User` Table is provided by the authentication framework, and it is best left untouched. Any additional fields are to be added to `UserProfile` instead. At this point we did not actually identify any suitable additional field to attach to the profile (remember: "no frills"); the class is, nonetheless, a convenient place to store relevant methods. Incidentally, the recommendation to avoid modifying objects from the authentication framework is why GEEZMO does not use Django's implementation of user groups.

As a final remark, when GEEZMO decides to contact the manager, it sends a custom signal to `weather.problem_detected`. The code base provides a simple signal handler that sends an email to the supervisor; additional actions can be defined by creating and connecting new signal handlers in `signals.py`.

Whether we decide to raise an alarm or not, we should support the supervisor in helping decide whether to escalate the situation by starting talks with his manager. We need to anonymise data as much as possible to protect the anonymity of voters. Voters anonymity however can be turned against the system, through simple frequency analysis – and much more can be done to limit the negative effects thereof, for example through "fuzzing" techniques.

Frequency analysis however can also be a force for good, however, as it can uncover some use of anomalous voting patterns such as hostile voting. As a result the supervisor's reporting interface should show not simply one day's voting average and standard deviation, but actual separate counts for each possible vote or prediction. This should prove as an important data point for those evaluating the effectiveness of GEEZMO at measuring team morale in the real world.

Table 1: Estimated likelihood of a prediction "successfully" guessing a single vote when using a "distance from average" criterion [13].

i	$\alpha = 1.5$		$\alpha = 1$	
	r_i	$P(v_i \in r_i)$	r_i	$P(v_i \in r_i)$
1	$(1\ \ , 2.5)$	29%	$(1, 2)$	14%
2	$(1\ \ , 3.5)$	67%	$(1, 3)$	48%
3	$(1.5, 4.5)$	86%	$(2, 4)$	68%
4	$(2.5, 5\ \ )$	67%	$(3, 5)$	48%
5	$(3.5, 5\ \ )$	29%	$(4, 5)$	14%

4. PRELIMINARY VALIDATION: ARE PREDICTIONS "SUCCESSFUL"?

To get an idea for the validity of GEEZMO in practice we used statistical analysis and simulation of a development network composed of 100 voters. This section outlines details of our statistical analysis and the considerations we used for the evaluation of GEEZMO.

Let n voters cast their votes $v_i \in \{1, \dots, 5\}$ anonymously, with 1 being the most negative response and 5 being the most positive. Predictors can also choose $p_j \in \{1, \dots, 5\}$.

We need to define somehow when a prediction p_j is sufficiently close to that day's votes $\mathbf{v}$ to be considered "successful" – and more importantly when it is not and there is cause for concern. Let us consider three approaches to achieve this goal described in the following.

4.1 Distance from Average

Results for this approach are shown on Table 1. Let $\bar{v}$ be the average of votes. With this straightforward approach, the j-th predictor is successful if $|p_j - \bar{v}| < \alpha$, for some value of α that is lax enough to reduce false alarms (for example $\alpha > 1$) but tight enough to allow the possibility of failure (for example $\alpha < 2$). For example, it might seem reasonable to pick $\alpha = 1.5$.

Let us call the range of values $r_j = \left(r_j^-, r_j^+\right)$ of $\bar{v}$ that makes a choice p_j of prediction successful the prediction's "range". This method is not very effective because, effectively, $r_5 \subset r_4$. Indeed, since we had chosen $\alpha < 1$, $r_5^- = 5 - \alpha < 4$ and $r_4^+ = 4 + \alpha > 5$. Perhaps more concerningly, r_3 covers almost the entirety of the possible range of values for $\bar{v}$: if $\alpha = 1.5$, then $r_3 = (1.5, 4.5)$.

Assume, for clarity's sake, that $v_i \simeq \mathcal{N}(3, 1)$[2] are independent (although they clearly are not) and are identically distributed. Even with a "restrictive" choice of $\alpha = 1$, $P(v_i \in r_3) = P(2 < v_i < 4) \simeq 68\%$; with the original choice of $\alpha = 1.5$,$P(v_i \in r_3) > 86\%$. These odds are unacceptably high, especially when compared to the odds of the other choices.

Things get only worse when we compare the predictions against the actual average $\bar{v} = \frac{1}{N}\sum_i v_i \simeq \mathcal{N}\left(3, \frac{1}{N}\right)$: as the number of participants increases, the variance of $\bar{v}$ decreases and thus the likelihood of $p_j = 3$ being classified as successful grows even further.

Of course, v_i's are not independent. Given two jointly normally distributed random variables $X, Y \simeq \mathcal{N}\left(\mu, \sigma^2\right)$ with correlation ρ, it can be shown that $\frac{X+Y}{2} \simeq \mathcal{N}\left(\mu, \frac{\sigma^2(1+\rho)}{2}\right)$;

[2] $v \simeq \mathcal{N}\left(\mu, \sigma^2\right) \implies P(v \in [\mu - 2\sigma, \mu + 2\sigma]) > 95\%$. By choosing $\mu = 3$ and $\mu - 2\sigma = 1$, the choice of $\sigma = 1$ follows.

since we expect $0 > \rho > 1$, the combined variance increases and the situation is somewhat ameliorated. This is but a pyrrhic victory, unfortunately: in the best case scenario, $\rho = 1$, we have $X \equiv Y \equiv \frac{X+Y}{2} \simeq \mathcal{N}(\mu, \sigma)$, and successfully predicting the average of two votes becomes no harder than predicting a single vote.

4.2 Segmented Average

A much better approach could be obtained working backwards from the previous analysis: what if we made each range r_i equally likely? For example, we could consider this simple "two options" rule: the j-th predictor is successful if $p_j = \{1, 2, 3\} \wedge \text{avg}(\mathbf{v}) < 3$ ("bad day") or if $p_j \in \{4, 5\} \wedge \text{avg}(\mathbf{v}) > 3$ ("good day"). It is pretty obvious here that, assuming the initial assumption $v_i \simeq \mathcal{N}(3, 1)$ holds true, random choice will not result in consistent success.

Since the ranges in "two options" are however very wide, we can also consider what happens when we divide the 1-5 range in more equiprobable ranges, separated by the 20th, 40th, 60th and 80th percentiles of $\mathcal{N}(3, 1)$. The results of doing so are shown in approximate form in Table table 2 under the "five options" and "four options" column. In "five options" there is no overlap between ranges and each is theoretically 20% likely to be the correct one. "Four options" obtains its four ranges through pairwise concatenation of ranges from "five options", obtaining overlapping ranges each 40% likely to be correct; a fifth "do not know" option is also included.

4.3 Poll Satisficing and the Midpoint Option

Poll theory points to the phenomenon of poll satisficing [14]: when given a bipolar scale rating with a midpoint, such as in this case, the gives the appearance of being "the easy way out:" the middle option has been shown to be the easiest to defend and, while it takes little cognitive load to choose it, it also gives the impression of careful choice.

Since $3 \in r_3$ for all methods considered so far, picking this option essentially means "this was an average day, neither good or bad." This is probably true of most days, however. If we deny the midpoint, as it was done in the "bad day"/"good day" example, we force predictors to take sides.

As an aside, this might rise the question: why offer the midpoint to voters, then? Would not voters also mindlessly pick this middle option? This seems likely. However, we must not lose sight of the greater context here: voters are developers and their job description is implementing software, not voting on polls. One of the design goals here is to minimize the context switch penalty that daily polls may introduce. Predictors, on the other hand, are managers who are supposed to keep up with and facilitate their team's operations. Making their polling work slightly harder and forcing them to take sides does not therefore seem unfairly taxing.

That does not mean that we should not anticipate and compensate for this effect. One way of doing so is through the option labels: we choose "`:-|`" for $v_i = 3$, which has (in the author's opinion) a slightly negative connotation; an "average" state of mind would rather be described through the smiley face "`:-)`", chosen for option 4. Another way we can account for it is by adjusting the segments further.

This phenomenon is the rationale behind devising the "four options" approach. The midpoint prediction option is still available in the interface, but it is interpreted as a "do not know" response and is not registered. This discards attempts at mindless poll satisficing. Additionally, adjacent intervals overlap, granting predictors a little "grace" interval and reducing the chance of false negatives. Finally, random choice of p_j fares worse than a coin flip.

This analysis might superficially make "four options" the clear winner. However, there are a number of other important issues with it that we will now discuss.

Table 3: An example of GEEZMO configured to support hierarchical structures recursively.

Team	Predictor	Voters	Supervisor
1	C	D, E	B
2	G	H, I	B
3	B	C, F, G	A
4	J	K, L	A
5	B, J, A		A

(Teams 1–4 are hierarchical. Team 5 is circular.)

5. DISCUSSION

5.1 Beyond Hierarchical Social Structures

The presence of hierarchy might be taken for granted in some, but not all, social structures found in software development. Our previous work [24, 23], identifies 13 different types of organizational social structures commonly found in the wild.

In particular, some community types do not have sufficiently strong hierarchies to support classical problem reporting and solving approaches, for example:

- Informal Communities
- Problem Solving Communities
- Informal Networks

These communities present strong egalitarian characteristics and thrive without a strategic hierarchy for typical mood-sensing strategies to exploit. To address this limitation a variant on the "hierarchical team" model in our research solution was be the "circular team" model, where all voters can also be predictors. The consequent challenge we still face is the need to identify one team owner who will act as the supervisor, but otherwise every member act as both somebody who submits information about their morale and somebody who tries to maintain others' morale high. Table 3 recaps organisational social structure types arranged by circular or hierarchical logic.

5.2 Hostile Voting

In our simulations we observed that hostile voting can have at least a 40% chance of changing the predictions outcome – and if the outcome does change, it does in the direction of the hostile voter's random choice of 1 or 5 for that day. Here we see the reverse outcome from the previous analysis: two options virtually guarantees the hostile voter a successful prediction, whereas "five options" and "$\alpha = 1$" perform significantly better. In particular, when comparing their measured 36% success rate with the respective theoretical success rates of 40% and 14%, "five options" is the only solution that performs acceptably.

Table 2: Approximate $\bar{v}$ ranges of "successful" predictions derived from the "segmented average" approach [13].

i	Two options		Five options		Four options	
	$P(v_i \in r_i)$	r_i	$P(v_i \in r_i)$	r_i	$P(v_i \in r_i)$	r_i
1	50%	(1, 3)	20%	(1 , 2.2)	40%	(1 , 2.8)
2			20%	(2.2, 2.8)	40%	(2.2, 3.2)
3			20%	(2.8, 3.2)	(no vote)	
4	50%	(3, 5)	20%	(3.2, 3.8)	40%	(2.8, 3.8)
5			20%	(3.8, 5)	40%	(3.2, 5.0)

Things get even more interesting when more than one people adopt this strategy, however, and this is where "five options" really shines. As game theory tells us, if a strategy is optimal, *all* rational players will use it. If we were to apply gamification to such a system, this would give further incentive to reap gamification's rewards as opposed to truthfully answering the question.

In order to analyze this, we ran another simulation: we analyzed what happens when 100 people vote and an increasing number h of them engages in hostile voting, whereas the rest votes like $\mathcal{N}(3, 1)$. "Two options" unequivocally rewards hostile voting: even as everybody rolls their 1-or-5 vote independently, the chances of getting a correct predictions increase beyond 50% as more people engage in hostile voting[3]. On the other hand, the other methods give additional hostile voters diminishing returns: in particular, you are always more likely to "get it right" if you vote honestly, and "five options" again shines as the solution that performs the best.

5.3 "Everything is Terrible" (or Perfect)

That said, that does not mean that there are no very simple strategy that leads to every voter/predictor being 100% correct 100% of the time. For example, everybody could agree to always pick 1 (a strategy we affectionately call "everything is terrible"). Similarly, everybody could agree to say "everything is perfect" and always pick 5.

There can be no real solution to this without external intervention (from reprimanding the relevant teams to simply giving up). The best we can do is not to encourage people (even if inadvertently) to adopt such a strategy in the long run. This is the main reason why we conjecture that applying gamification to GEEZMO could be counterproductive. More discussions on this conjecture are provided in the next Section.

5.4 Gamification with GEEZMO

Gamification techniques are based in creating a reward and incentive structure in the use of the software. Made originally popular by Stack Overflow, gamification spread quickly to other "social" software with varying degrees of success. The main issue when designing such a system is making it so that attempting to "game the system" in order to reap as many benefits from the system as quickly as possible pushes the users towards desirable behaviour.

For example, Stack Overflow uses two different benefit systems: reputation, which is awarded by other users for your performance, and badges, which are usually awarded by the site for performing a task for the first time (tasks such as "filling out your profile" or "voting an answer down – any answer").

The three main (and essentially only) interactions GEEZMO has are voting, predicting and supervising. We now present a number of badges we could award for each, and why implementing them would be a bad idea:

- "First vote!" (voted for the first time)

  Awarding this badge hampers the anonymity characteristics of GEEZMO. If you gain access to the list of anonymized votes and compare the vote history before and after this badge was awarded, it is possible for a third party to extrapolate how the newcomer is voting. This could also encourage the newcomer to vote something – *anything* – although chances are that a new user would not be savvy to a reward structure built into the website they just used for the first time.

- "Returning customer" (voted n times)

  If, however, the user is awarded a badge with great fanfare for voting for the first time, chances are they will look at the list of badges, and "vote n times"-style badges do nothing but get the user to vote on something – *anything* – for a number of days. This is especially problematic if the badge requires voting for n days in a row. What we would like to have is people voting honestly. What this badge risks actively encouraging is mindless voting for the midpoint.

  Worse still: in order to actually implement such a badge, you'd have to maintain at the very least a counter in your database saying how many votes each user has cast. This gives an attacker more information when performing frequency analysis.

- "Oh, happy days!" (voted 5 for the first time), "I need a hug" (voted 1 for the first time), "Jack of all Moods" (voted for each mood at least once), etc.

  Again, such badges provide unacceptable leaks in anonymity and very strong information in backwards engineering of one user's voting patterns in case those badges have not been awarded.

- "Observer" (predicted for the first time)

  In the hierarchical model of teams, where there is exactly one predictor – the manager – predictions are not

[3]This is perhaps counter-intuitive. If 100 voters/predictors engage in hostile voting using the two options system, the system degenerates in a game where 100 people flip a coin at the same time and the result that is more common in each "round" is also the winning one. If there are 50 tails and 50 heads, everybody wins. If there are 49 heads, there are 51 tails, and thus 51 winners – and viceversa. As a result, your winning chances in this game are strictly greater than 50%.

anonymous and he or she has not much they can do to affect the outcome of voting, short of flat out telling people what to pick. In a way, there is no strong reason not to award this particular badge in this scenario.

Things are not so true in the circular system, as previously explained. When you vote and predict, the best strategy is making your prediction and your vote the same. Even if we allowed people to make separate predictions from their votes, we would essentially be encouraging them to vote publicly – and one of the design goals is to encourage the opposite: surfacing unvoiced dissent. So, in circular teams, we must also have anonymous predictions. This makes that badge no different than "First vote!".

- "Fortune teller" (first prediction scored as "successful"), "Oracle" (scored n successful predictions), etc., or otherwise getting points for successful predictions.

  In the hierarchical model of teams, we clearly want to have managers that predict well, keep up on what's going on and enable their team to work productively. However, we do not want to encourage that so much that managers get in a competition with one another. A manager could simply "cheat" by be peeking over people's shoulders as the time to voting comes up. Of course, this puts unwanted pressure on people who will face the same pressures in their anonymous vote as they would if they spoke out in the open – defeating the purpose of the exercise. This is why it is safer not to tell the manager if and when he makes "successful" predictions: after all, exceptional skill in prediction does not necessarily map to successfully managing a team.

  In the circular model of teams, such badges would essentially be disastrous as they encourage and reward "everything is terrible"-style voting.

- "Condition green" (supervisor gets an alarm for the first time)

  In the case of highly hierarchical structures, a supervisor does not have such a role for a large number of teams. If such a badge was "awarded", voters would notice and start speculating on who's voting what, creating interest and gossiping that could very well work against the successful resolution of a situation. Even more critically, we think it's critically important to allow supervisors to receive an alert, study the situation and decide that it was probably a false alarm, a voting anomaly or that otherwise there is not enough evidence of unsurfaced unrest to unnecessarily unnerve an unsuspecting manager.

As a consequence attempts to gamify GEEZMO at best do not help and at worst directly work against its design tenets and goals. More research should be invested to clarify this research questions since gamification techniques have seen successful usage in the past, e.g., to assess software development progress or increase software development awareness [25].

6. RELATED WORK

Polling is now a very mature field of science, and much ink has been spilled on the subject.

Scholar research on repeated administration of the same poll is somewhat more sparse, and generally focused on correctly aggregating responses collected at separate points in time in order to *remove* the effects of time [7, 1, 19].

One attempt in exploiting the time dimension is instead proposed in the Delphi technique [3] that, for example, tries to administer the same comprehensive poll over and over with a basic response summary being provided to participants between each round.

The goal is to reach agreement between experts on a given topic in a pressure-free environment, as responses are anonymous. It appears, however, that the feedback loop provided by the response summary might be strong enough to cause GEEZMO to be self-defeating by creating pressure towards conformity [20].

Our research solution however is more closely related to the field of sentiment analysis or opinion mining. Indeed, much of said work stems from the MSR community [8]. For example, meta-algorithms that analyze sentiment from multiple bodies of text and classify them on a scale from 1 to 4 have been shown to work [17]. Work has also been done to classify shorter text snippets such as tweets according to a three-way polarity scale (positive/neutral/negative) [12].

However, we could not find any research about automated sentiment analysis of video or voice feeds (which could, conceivably, be based on simpler metrics than natural language processing uses); as a result, they would not be able to process all team communication (never mind the privacy consequences such a process would entail).

7. CONCLUSIONS

This paper elaborates on a High-Frequency Mood-Polling for Software Development "Weather Prediction" across software projects. Our research assumption is that using non-invasive mood-polling may be helpful in predicting sub-optimal organisational or social patterns taking place across a software development community. This paper also outlines our proof-of-concept prototype and how it was validated using simulations and statistical analysis methods.

Our preliminary validation shows interesting results into a tentative strategy at supporting team awareness on social and organisational issues across the organisational structure. I the future we plan to refine our prototype and deploy it in a practical experiment to evaluate its actual potential.

Also, we plan to further explore the limitations we currently evaluated for GEEZMO in terms of its fitness for usage within the scope of Gamification campaigns. We have observed there are severa limitations that might be overcome by investing additional research, e.g., investigating GEEZMO typical usage patterns and understand how they might be used as a basis for gamification.

8. ACKNOWLEDGEMENTS

The authors' work is partially supported by the European Commission grant no. 610531 (FP7 ICT Call 10), SeaClouds.

9. REFERENCES

[1] B. Blight and A. Scott. A stochastic model for repeated surveys. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 61–66, 1973.

[2] L. Bock. Passion, not perks. Thinking with Google, September 2011.

[3] N. C. Dalkey. Delphi. 1967.

[4] I. Epifani. Verifica di ipotesi. Appunti delle lezioni del corso di Statistica (2L) per gli allievi INF e TEL, AA 2008/2009, June 2009.

[5] S. Exchange. Company page: Stack exchange. Careers Stack Overflow, December 2014.

[6] S. Exchange. Stack exchange open source projects. Stack Exchange Blog, December 2014.

[7] M. Feder. Time series analysis of repeated surveys: The state–space approach. *Statistica Neerlandica*, 55(2):182–199, 2001.

[8] E. Guzman, D. Azócar, and Y. Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 352–355, New York, NY, USA, 2014. ACM.

[9] W. Heisenberg. Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. *Zeitschrift für Physik*, 43(3-4):172–198, 1927.

[10] S. Invernizzi and S. Gatti. Adding awareness to a software forge: development of the titans approach and lessons learnt in participating in an apache open source development process. 2013.

[11] V. A. Kotelnikov. On the carrying capacity of the ether and wire in telecommunications. In *Material for the First All-Union Conference on Questions of Communication, Izd. Red. Upr. Svyazi RKKA, Moscow*, 1933.

[12] E. Kouloumpis, T. Wilson, and J. Moore. Twitter sentiment analysis: The good the bad and the omg! *ICWSM*, 11:538–541, 2011.

[13] W. A. LLC. Wolfram|alpha, November 2014.

[14] L. E. Lyberg, P. Biemer, M. Collins, E. D. De Leeuw, C. Dippo, N. Schwarz, and D. Trewin. *Survey measurement and process quality*, volume 999. John Wiley & Sons, 2012.

[15] J. A. Nelder and R. Baker. *Generalized linear models*. Wiley Online Library, 1972.

[16] J. Neyman and E. S. Pearson. *On the problem of the most efficient tests of statistical hypotheses*. Springer, 1992.

[17] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL*, pages 115–124, 2005.

[18] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[19] D. Pfeffermann. Estimation and seasonal adjustment of population means using data from repeated surveys. *Journal of Business & Economic Statistics*, 9(2):163–175, 1991.

[20] G. Rowe and G. Wright. The delphi technique as a forecasting tool: issues and analysis. *International journal of forecasting*, 15(4):353–375, 1999.

[21] D. Tamburri, P. Kruchten, P. Lago, and H. van Vliet. What is social debt in software engineering? In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pages 93–96, May 2013.

[22] D. A. Tamburri, E. di Nitto, P. Lago, and H. van Vliet. On the nature of the GSE organizational social structure: an empirical study. *7th International Conference on Global Software Engineering*, pages 114–123, 2012.

[23] D. A. Tamburri, P. Lago, and H. van Vliet. Uncovering latent social communities in software development. *IEEE Software - Special Issue on Bridging Software Communities through Social Networking*, pages 26–32, 2012.

[24] D. A. Tamburri, P. Lago, and H. v. Vliet. Organizational social structures for software engineering. *ACM Computing Surveys (CSUR)*, 46(1):3, 2013.

[25] B. Vasilescu. Human aspects, gamification, and social media in collaborative software engineering. In P. Jalote, L. C. Briand, and A. van der Hoek, editors, *ICSE Companion*, pages 646–649. ACM, 2014.