Prisoner's Dilemma in Software Testing

Loe Feijs
Eindhoven University of Technology

Abstract

In this article the problem of software testing is modeled as a formal strategic game. It is found that for certain values of the productivity and reward parameters the game is essentially equivalent to the Prisoner's Dilemma. This means that the game has a unique Nash equilibrium, which is not optimal for both players, however. Two formal games are described and analyzed in detail, both capturing certain (though not all) aspects of real software testing procedures. Some of the literature on the Prisoner's Dilemma is reviewed and the results are translated to the context of software testing.

Keywords: software testing, software quality, game theory, Nash equilibrium, prisoner's dilemma.

1 Introduction

Software is playing an increasingly important role in modern society. Not only in office software and computer games, but also in embedded systems such as in televisions, telephony exchanges, cars, etc. computer programs of considerable size are essential. Software bugs constitute a serious problem [1]. Software testing is frequently used to find errors so they can be repaired and hopefully, the software quality improved. In view of the societal relevance of software quality, it makes sense to consider software quality as an economic issue and to use rational methods when studying it. In this article we use concepts from game theory for that purpose. The problem of software testing is modeled as a formal strategic game. One of the findings of this article is that this game is of a special nature, known as Prisoner's Dilemma [2].

The article is structured as follows. Sect. 2 introduces the necessary game theory, which includes the Prisoner's Dilemma. Sect. 3 describes the role of testing in software engineering. Sect. 4 describes an idealized testing game and Sect. 5 discusses a subtle variation of the same game. We have an example of a concrete testing game, providing support for and adding plausibility to the game of Sect. 4. Its presentation is very technical, however; therefore all the details are described in App. A. In App. B this concrete testing game is analyzed. In Sect. 6 the effect of multiple performance levels is investigated. In Sect. 7 several known results on the Prisoner's Dilemma are summarized and translated to the context of software testing. Finally Sect. 8 contains some concluding remarks.

2 Prisoner's Dilemma

There are two types of games that are studied in game theory: zero sum games and non-zero sum games. Zero sum games obey the rule that the sum of the payoffs to all players equals zero. An example is the game of chess where a player receives one point if he wins, -1 point if he looses and 0 points in case of a draw. There are two players and three outcomes: white wins and black looses, white looses and black wins, and a draw (no one wins). The payoffs are $\langle 1, -1\rangle$, $\langle -1, 1\rangle$ and $\langle 0, 0\rangle$, respectively. In each case, the sum equals zero, e.g. if white wins the payoffs are given as the pair $\langle 1, -1\rangle$ so the sum $1 + (-1) = 0$. Zero sum games were studied by Zermelo and Von Neumann [3] who established solution concepts and proved their

existence using minimax analysis and fixed point theories. Zero sum games model situations where players have a conflict of interests such that their interests are completely opposite.

Non-zero sum games, by contrast, model situations in which there is a conflict of interest, but where the opposition is not complete. To a certain extent, the players can have shared interests, but their interests are not completely aligned either. Throughout this article the class of non-zero sum games is needed.

Another distinction usually made in game theory is between strategic games and extensive games. In a strategic game all players choose their actions once in a simultaneous fashion whereas an extensive game is more general; in an extensive game it is possible to perform several actions in a sequential fashion (as for example in chess and checkers). Throughout this paper we focus on strategic games, which are formally introduced next.

A strategic game is a triple $\langle N, (A_i), (u_i) \rangle$ where N is a finite set of players and for each player $i \in N$ there is a set A_i of so-called actions. For each player $i \in N$ there is a function $u_i : A \rightarrow \mathbb{R}$ that assigns a payoff to each tuple of actions. Here $A = \Pi_{i \in N} A_i$, or in the special case that the size of N equals two, $A = A_1 \times A_2$. The payoffs give rise to a preference ordering $\geq_i$ on A, one ordering for each player, defined by $a \geq_i b$ iff $u_i(a) \geq u_i(b)$. The absolute values of the payoffs are considered irrelevant in the sense that only the induced preference ordering counts. Throughout this article the number of players equals two. Thus N contains two elements and without loss of generality it can be assumed that $N = \{1, 2\}$. A pair of actions $\langle a, b \rangle$ for $a \in A_1$ and $b \in A_2$ is called an action profile.

A simple example of a strategic game is BoS, the Battle of the Sexes, in [5] explained as Bach or Stravinsky. It is here used as an example to introduce a convenient tabular notation called payoff matrix. The actions of player 1 correspond to the rows of the matrix. The actions of player 2 correspond to the columns of the matrix. Each entry in the matrix contains a pair: the payoff of player one, followed by the payoff of player 2. The payoff matrix of BoS is:

	B	S
B	2,1	0,0
S	0,0	1,2

The action sets are $A_1 = A_2 = \{B, S\}$, that is, each player has to indicate his choice for a concert of music, going to either Bach (B) or Stravinsky (S). Their main concern is to go together, but one person prefers Bach and the other prefers Stravinsky.

The interesting question is which action profiles are to be chosen by rational players. A very important concept for studying this question is the Nash equilibrium (the original publication is [4]; we follow the presentation of [5]). A Nash equilibrium (N.E.) of a strategic game $\langle N, (A_i), (u_i) \rangle$ for $N = \{1, 2\}$ is an action profile $\langle a^*, b^* \rangle$ with the property that $\langle a^*, b^* \rangle \geq_1 \langle a, b^* \rangle$ for all $a \in A_1$ and similarly $\langle a^*, b^* \rangle \geq_2 \langle a^*, b \rangle$ for all $b \in A_2$. The intuition is that a Nash equilibrium is a consistent expectation pair in the sense that player 1, if he is rational, sticks to his choice a^* which is better for him than any other choice (assuming player 2 does not deviate from his b^*) and conversely for player 2. The concept of Nash equilibrium is illustrated by the game of BoS, which has two Nash equilibria, viz. $\langle B, B \rangle$ and $\langle S, S \rangle$. The preferences are shown in the diagram of Figure 1 which is like the payoff matrix except for the arrows which indicate how player 1 can improve a certain action profile by another action profile (according to the vertical arrows) and similarly for player 2 (according to the horizontal arrows).

After these preparations the Prisoner's Dilemma is introduced. The Prisoner's Dilemma is the two-player strategic game with the following payoff-matrix:

	C	D
C	−1,−1	−3,0
D	0,−3	−2,−2

The game was proposed by Merril Flood and Melvin Dresher in 1950 in a slightly different setting. The name 'Prisoner's Dilemma' was proposed by Albert Tucker, who found the anecdote of the two prisoners. In [2] the game is explained as follows:

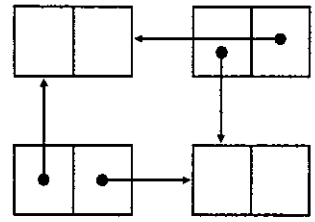

Figure 1: Improvements for the BoS game.

> Two members of a criminal gang are arrested and emprisoned. Each prisoner is in solitary confinement with no means of speaking to or exchanging messages with the other. The police admit they don't have enough evidence to convict the pair on the principal charge. They plan to sentence both to a year of prison on a lesser charge. Simultaneously, the police offer each prisoner a Faustian bargain. If he testifies against his partner, he will go free while his partner will get three years in prison on the main charge. Oh, yes, there is a catch ... If *both* prisoners testify against each other, both will be sentenced to two years in jail.

This explanation motivates the sets $A_1 = A_2 = \{C, D\}$ where D means 'Defect' and C means 'Cooperate'. Since the absolute values are irrelevant, the Prisoner's Dilemma can equally well be described by the following payoff matrix (taken from [5]):

	C	D
C	3,3	0,4
D	4,0	1,1

There is something paradoxical about the Prisoner's Dilemma which is the fact that the only N.E., the action profile $\langle D, D\rangle$ with payoff pair $\langle 1, 1\rangle$ is inferior to the pair $\langle C, C\rangle$ with payoff pair $\langle 3, 3\rangle$. It is inferior for both players. The improvements diagram of Figure 2 demonstrates that $\langle D, D\rangle$ is the only N.E.:

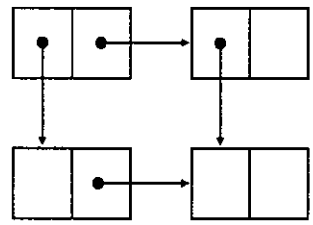

Figure 2: Improvements for the Prisoner's Dilemma.

3 Software testing

Since the invention of stored-program computers in the 1940s the importance of computer programs has been steadily increasing. Today, many millions of people rely on software such as Unix, Windows, MS Office for important aspects of their work. Televisions, telephony exchanges, cars, airplanes, mobile phones, etc. each contain thousands, sometimes millions of lines of embedded code. Experience shows that it is very hard to guarantee the correctness of these programs (often referred to as software). Sometimes the problems seem out of control, as illustrated by the problems reported in [1]: Software's Chronic Crisis. Gibbs mentions the

example of the baggage-handling system of Denver's new international airport that, for nine months, was held captive by "Lilliputians-errors in the control software". Shaw of Carnegy Mellon, Bourgonjon of Philips and others note how software is exploding. Gibbs writes about bugs, small glitches, the density of errors and so on. Software testing is used in serious attempts to improve the situation by detecting errors as soon as possible. As Hall puts it in [1]: "The benefits of finding errors at that early stage is enormous". Testing plays an important role in the present industrial practice, particularly when the software systems are very large. As an example we mention the development of Windows 2000, being 29 million lines of code. We quote from [6] where it is reported that: "[ ] at the level of size and complexity of Windows 2000, writing code was no longer the central activity. Indeed, testing and debugging have accounted for between 90 and 95 percent of the work". It is generally believed that large-scale software projects have to spend 50% or more of their effort in testing.

There is a large literature on the art and science of software testing. Formal method specialists stress the fact that testing can reveal errors but can not prove the absence of errors (Dijkstra EWD340); both research and case studies are done to explore the power of correctness proofs and automated analysis tools (model checkers). Despite these research efforts, most practitioners consider some form of testing indispensible. In the field of protocol conformance testing, the topic of testing is subject of much research and development. We mention the language TTCN [7] and Finite State Machine based methods such as those mentioned in [8], [9] and Labeled Transition System methods, for example [10].

Most authors in formal testing research adopt the viewpoint that there should be a formal specification which is used as a starting point for procedures to generate tests, execute them, and determine their coverage. The main goal seems to be that the efficiency, or error detection power of the tests-to-be-derived is optimized.

Useful as this may be, little or no attention is given to the fact that there is an opposition of interests between the tester and the implementor. The goal of the tester is to find errors, whereas it is easily observed in practice that an implementer likes his program to be shown correct (as he often believes it is). The practical advise to separate these roles, letting the tester be another person than the implementer is well-known for this reason.

In this article we do not just seek an optimizing procedure for the tester, but we focus on the opposition of interests. Since game theory has a vocabulary and analysis methods for situations of opposite interests, these can be applied to software testing. This is undertaken in the next section.

4 An idealized testing game

In this section a very simple and yet fairly general model is proposed first. The model is nothing but a payoff matrix for two players: an implementer (player 1) and a tester (player 2). After that the model will be refined to bring in the probabilistic aspects that ocur when neither the implementation nor the test are perfect (so both PASS and FAIL are possible outcomes, each with its own probability). The model is abstract in the sense that it does not tell what an implementation or a test looks like. A more concrete example model, showing that the abstract model makes sense, is postponed until App. A.

The Idealized Testing Game (ITG) is the two-player strategic game $\langle N, (A_i), (u_i) \rangle$ where $N = \{1, 2\}$ and $A_1 = A_2 = \{p, q\}$, with the intuition that p means 'poor' (bad) and q means 'quality' (good). The u_i are given by the payoff matrix:

	p	q
p	2,2	0,3
q	3,0	1,1

The motivation for this payoff matrix is as follows. The implementer is rewarded if he delivers an error-free, or almost error-free piece of software. The tester is rewarded if he performs a thorough testing job. In practice it is hard to tell whether the software is error-free or not;

that is precisely what testing is meant for. Therefore in ITG the implementer is rewarded if no error is found (an outcome denoted as PASS). The tester should be eager to find errors. Therefore in ITG, the tester is rewarded if at least one error is found (an outcomed denoted as FAIL). The implementer chooses between doing a poor job, delivering software containing errors, and doing a quality job, delivering error-free or almost error-free software. The tester chooses between doing a poor job, performing a test with low error detection capability, and performing a quality test, which takes more effort, but which is more likely to reveal errors. The payoffs for the action profile $\langle p, p\rangle$ are 2 for the implementer and 2 for the tester. Both do a poor job, so yes, the software contains errors, but at the same time there is a low error detection capability by the tester's choice. Some of the errors are found, some are not. Both the implementer and the tester are equally well rewarded; this explains both payoffs that are 2 (recall that the absolute value is irrelevant). Another situation occurs when an almost perfect implementation is investigated in an almost perfect test. But now the payoffs are 1 since both players have to make serious investments in effort (time), and perhaps also in costs for more sophisticated personal education, tools etc. The costs of these efforts, here estimated to have a value of 1, are to be subtracted from the rewards, both for the implementer and the tester.

Next consider the action profile $\langle q, p\rangle$, that is, an almost perfect piece of software investigated by a poor test. Clearly this gives a PASS. The implementer's reward increases from an average of 2 to a certain value of 4. But the effort is higher too, so the implementer gets 3. The tester has no costs for the effort, but no rewards either. The effect of reward outperforms the effort (=1) so the tester's payoff is 2 less than his payoff for $\langle p, p\rangle$; the tester gets 0. Conversely, $\langle p, q\rangle$ gives a FAIL with payoffs 0 and 3.

The payoff matrix of ITG is the payoff matrix of a Prisoner's Dilemma. There is one N.E., the action profile $\langle q, q\rangle$. In the N.E. both players choose to do a quality job.

Next this model is to be refined in order to bring in the probabilistic aspects. The refined model is called ITG'. It is defined by a 4×4 payoff matrix. From ITG' the earlier model ITG can be derived in a formal way: the four choices for the implementer are reduced to two choices, each of which is a lottery over two alternative implementations of equal quality level (and similarly for the other player). By this approach, the numbers from the ITG payoff matrix can be retrieved later (in other words: ITG' will provide a more detailed motivation for ITG).

The refined Idealized Testing Game (ITG') is the two-player strategic game $\langle N, (A_i'), (u_i')\rangle$ where $N = \{1, 2\}$ as before and where $A_1' = A_2' = \{p1, p2, q1, q2\}$. The u_i' are given by the payoff matrix:

	p1	p2	q1	q2
p1	4,0	0,4	0,3	0,3
p2	0,4	4,0	0,3	0,3
q1	3,0	3,0	3,−1	−1,3
q2	3,0	3,0	−1, 3	3,−1

The motivation for this payoff matrix is as follows. The implementer has always several alternative design choices, even after he has made a conscious or unconscious decision to spend the effort corresponding to a poor job or the effort corresponding to a quality job. In real life this design space is huge but here it is assumed to be a two-element set. It is the set $\{p1, p2\}$ for 'poor' and the set $\{q1, q2\}$ for 'quality'. The reward for a programmer doing a good job, as witnessed by a PASS, equals 4 (think of 4$, 4K$ or 400K$ depending on the size and complexity of the specification). In the case of a FAIL the programmer receives reward 0. The tester gets 4 for a FAIL, 0 otherwise. A poor implementation and a poor test cost 0. A quality implementation costs 1 (the effort), which is counted negatively. The same holds for the test cost. It is assumed that $\langle p1, p1\rangle$ and $\langle p2, p2\rangle$ produce a PASS whereas $\langle p1, p2\rangle$ and $\langle p2, p1\rangle$ produce a FAIL. But if the programmer chooses *any* of his quality alternatives $q1$ or $q2$ he enforces a PASS whenever the tester chooses one of the poor alternatives. And so on, as conveniently summarized by the following matrix (call this a verdict matrix):

	p1	p2	q1	q2
p1	PASS	FAIL	FAIL	FAIL
p2	FAIL	PASS	FAIL	FAIL
q1	PASS	PASS	PASS	FAIL
q2	PASS	PASS	FAIL	PASS

This concludes the motivation of ITG'.

This game ITG' has no N.E. For example, consider the action profile $\langle p1, p1\rangle$ having payoffs 4,0. Then the tester can improve by choosing $p2$ instead of $p1$. The resulting action profile $\langle p1, p2\rangle$ has payoffs 0,4. Then the implementer can improve by choosing $p2$. The action profile $\langle p2, p2\rangle$ yields 4,0 so the tester improves to $\langle p2, p1\rangle$. The action profile $\langle p2, p1\rangle$ yields 0,4 so the implementer improves to $\langle p1, p1\rangle$. The 'improvements' go round in circles. Of course the reason is that the implementer has no reason to distinguish whether $p1$ or $p2$ is better because he has no way to know what the tester will do ($p1$ or $p2$).

Therefore the following two-step strategy is reasonable: first choose between p and q, then conduct a lottery over the alternative implementations. In reality an implementer does not feel like throwing dice concerning his implementation decisions; he tries his best with the selected time frame, but yet he makes mistakes at places where he is unaware of it and that is modeled as a random device.

In order to calculate the average payoffs, the payoffs are multiplied by a weighting factor. The sub-matrix where the implementer takes one of the p choices and the tester takes one of his p choices, contains four entries with equal probabilities. These must be 25%. The weighting matrix is:

	p1	p2	q1	q2
p1	25%	25%	25%	25%
p2	25%	25%	25%	25%
q1	25%	25%	25%	25%
q2	25%	25%	25%	25%

Performing an element-wise multiplication of the ITG' payoff matrix and the weighting matrix we get:

	p1	p2	q1	q2
p1	1,0	0,1	0,0.75	0,0.75
p2	0,1	1,0	0,0.75	0,0.75
q1	0.75,0	0.75,0	0.75,−0.25	−0.25,0.75
q2	0.75,0	0.75,0	−0.25,0.75	0.75,−0.25

Taking this latter matrix, the four weighted payoffs can be added for each of the four sub-matrices. For the $\langle p, p\rangle$ sub-matrix we must (player-wise) add 1,0 and 0,1 and 0,1 and 1,0 which means that the average payoffs for $\langle p, p\rangle$ are 2,2. The result is:

	p	q
p	2,2	0,3
q	3,0	1,1

This is easily recognized as ITG. So ITG comes out as an abstraction of ITG'.

Although ITG and ITG' are based on reasonable assumptions they are very abstract. The syntax and semantics of the programs and the details of the testing procedures are not elaborated and there is not even a difference between an implementer and a tester. We have an example of a concrete testing game TCTG (Text Copy Testing Game), providing support for and adding plausibility to ITG and ITG'. In TCTG the task for the implementer is to transcribe a given text, perhaps from one character set to another, or perhaps even simpler, to type precisely the characters of a given string. The length of the specification and the length of the

implementation are the same, L say. The task of the tester is to select one or more positions in the range 1 to L. If the implementation and the specification differ at one or more of the selected positions then an error is found and the verdict is FAIL. Otherwise the verdict is PASS. The presentation of the Text Copy Testing Game is very technical, however. Therefore all the details are described in App. A. In App. B this concrete testing game is analyzed.

Discussion: ITG is based on ITG′ which contains reasonable assumptions about the test process such as the assumption that a poor-poor confrontation and a quality-quality confrontation yield a 50%/50% ratio of PASS and FAIL. Another assumption is that the reward or success is 4 times the effort needed to perform a quality job. This factor of 4 is a parameter of the game (call it the reward/effort ratio, symbol R). If the ratio $R = 4$ then ITG results, essentially a Prisoner's Dilemma. Taking other values, other games result, some of which have another nature than the Prisoner's Dilemma. For arbitrary ratio R the adapted payoff matrix of ITG is:

	p	q
p	$\frac{1}{2}R,\frac{1}{2}R$	$0,R-1$
q	$R-1,0$	$\frac{1}{2}R-1,\frac{1}{2}R-1$

Its is interesting to consider values of R that are below 4; the nature of the game changes at $R = 2$. For example, taking $R = 1\frac{1}{2}$ the payoff matrix turns into:

	p	q
p	0.75,0.75	0,0.5
q	0.5,0	−0.25,−0.25

This is not a Prisoner's Dilemma. There is one N.E., viz. $\langle p, p\rangle$ and the corresponding payoffs 0.75,0.75 are better than the payoffs for any other action profile (for both players). If this is interpreted in the usual way by assuming that both player and tester behave rational and hence take this N.E., then it is a natural explanation to say that there is not enough incentive for both players to choose the quality alternative: the expectation of the reward does not outperform the effort.

5 What if the implementer moves first?

There is an assumption underlying the ITG that can be questioned, viz. that the implementer and the tester choose their performance level simultaneously. This is motivated by the way many software engineering projects are organised. As soon as the requirements analysis phase has been completed and the software specification is available, not only the implementer starts working, but also the tester. A large part of the tester's work consists of choosing test cases and carefully describing them. This work should not wait until the implementer is ready because of the usual requirement to keep the overall project duration as short as possible (another part of the work is test execution, which has to wait for the implementation anyhow). For the main development of the present paper we adopt the viewpoint that testing is a strategic game which means that both players move simultaneously, as motivated by the parallelism in the development project. It is also motivated by the fact that even after the implementer has delivered his implementation, the tester does not know the implementer's performance level (usually, testing is needed for that).

But, as a short side-line we shall briefly analyse a sequential version of ITG, wich we call ITG_{seq}. In ITG_{seq} the payoff matrix is:

	p	q
p	2,2	0,3
q	3,0	1,1

which is the same as for ITG. The difference is that in ITG_{seq} the implementer chooses first. Then, knowing the implementer's choice, the tester chooses. The motivation for this could be that in certain specific situations the tester can easily sample the implementer's work in order to estimate the performance level (for example if the code contains many deeply nested IF THEN ELSE statements, bugs are likely).

Next ITG_{seq} is analysed according to a max-maximization procedure (this is a two-phase optimization similar to the well-known min-max procedure for zero-sum games). If the implementer chooses p then the tester is left with the upper sub-matrix:

	p	q
p	2,2	0,3

The assumedly rational tester maximizes his own payoff and therefore chooses q. In this case the implementer receives payoff 0. Alternatively, if the implementer chooses q then the tester is left with the lower sub-matrix:

	p	q
q	3,0	1,1

The tester, maximizing his own payoff, chooses q. In this case the implementer receives payoff 1. The implementer is facing a choice between p with payoff 0 and q with payoff 1. Assuming the implementer is rational, he must choose q. Then the tester chooses q. So the solution of the max-maximization analysis is the (sequential) action profile $\langle q, q \rangle$. As it happens, this is the same as the N.E. of the ITG.

6 Adding an extra performance level

One of the objections one could have against the relevance of the ITG and the TCTG of App. A as a model of an implementer's behavior and a tester's behavior is the binary choice with respect to the performance level. Perhaps a real implementer does not want to choose between two extreme values of 'poor' and 'quality' levels, or between $Q = 20\%$ and $Q = 50\%$. A real implementer sees a whole range of performance levels, so perhaps he chooses between three levels (or even more). It is not a priori clear what implications this has for the nature of the game. ITG and TCTG make it difficult to choose between the poor level and the quality level (because of the paradoxical nature of the Prisoner's Dilemma) but will the dilemma disappear if there is a third, intermediate level? In order to investigate this, we construct another game called ITG3 which is a 3×3 strategic two-player game.

The game ITG3 is obtained from ITG by introducing an intermediate performance level halfway between 'poor' and 'quality'. The same notational devices used in App. B are used here. In order to simplify the calculations we approximate the effort values by linear interpolation (the rewards are always the same, but the effort part is varying). Therefore the refined payoff matrix is:

	p	pq	q
p	4,0 /0,4	4,−0.5 /0,3.5	4,−1/0,3
pq	3.5,0/−.5,4	3.5,−0.5/−0.5,3.5	3.5,−1/−0.5,3
q	3,0 /−1,4	3,−0.5 /−1,3.5	3,−1/−1,3

The weighting matrix is also taken to be the result of linear interpolation between the idealized values of P(FAIL) which are 0%, 50% and 100% in ITG. Therefore the weighting matrix is:

	p	pq	q
p	0.5 /0.5	0.25 / 0.75	0/1
pq	0.75/0.25	0.5 / 0.5	0.25/0.75
q	1/0	0.75 / 0.25	0.5 /0.5

The payoff matrix and the weighting matrix can be multiplied in an element-wise fashion to get:

	p	pq	q
p	2,0 /0,2	1,−0.125/0,2.625	0,0/0,3
pq	2.625,0/0.125,1	1.75,−0.25/−0.25,1.75	0.875,−0.25/−0.375,2.25
q	3,0/0,0	2.25,−0.375/−0.25,0.875	1.5,−0.5/−0.5,1.5

And by pair-wise adding the payoffs for PASS and FAIL the following payoff matrix is obtained (call this abstract 3×3 game ITG3):

	p	pq	q
p	2,2	1,2.5	0,3
pq	2.5,1	1.5,1.5	0.5,2
q	3,0	2,0.5	1,1

This is again a game with one N.E. at $\langle q, q \rangle$ but as in the Prisoner's Dilemma the payoffs 1,1 for the N.E. are less than certain other payoffs, such as the 2,2 for $\langle p, p \rangle$. The improvements diagram for ITG3 is shown in Fig. 3. From this analysis we see that introducing an intermediate

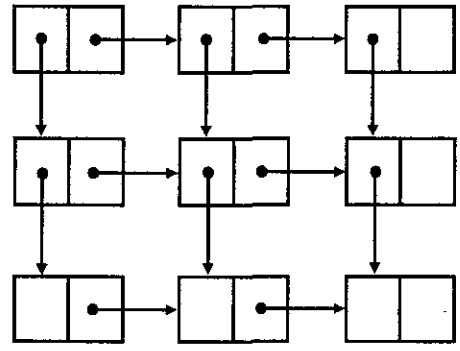

Figure 3: Improvements for the ITG3 game.

performance level does not change the essential nature of the game.

7 Testing is a Prisoners Dilemma; so what?

Consider a software development laboratory in which there are programmers (=implementers) and testers. The efforts and rewards are determined according to the principles such as those of ITG and TCTG. Programmers receive a fixed basis salary, but if they choose to do a 'quality' job, they have to spend more hours (which could have been spent otherwise in a profitable way, which means that the programmer pays for extra efforts). Moreover the programmer receives a bonus (the reward) if the subsequent test results in a PASS verdict. A similar payment system is adopted for the tester. There are other conceivable set-ups as well, for example introducing a programmer's boss who chooses the programmer's performance level and pays him accordingly; in this case it is the boss who plays the game, but it does not make much difference.

Now from the viewpoint of the software lab's manager: is it good or bad that there is a Prisoner's Dilemma going on in his lab? There are several answers possible, depending on the type of theoretical results that are employed and depending on the level of belief that these theoretical results apply to real life. Six different types of theoretical results are discussed:

- The concept of N.E. is a straightforward theoretical concept of 'solution' for a two player strategic game. If it is assumed that rational players choose the N.E. and if both the

programmer and the tester are rational, then they must choose the 'quality' performance levels; for the lab manager this means that they do their best to deliver quality software, which is the manager's (and the customer's) best outcome. It is best for the manager, not for the players. Looking at this way, ITG is good.

- Already in the early days of the development of game theory, serious doubt has been raised whether real people would behave rational in the sense of choosing the N.E., even in paradoxical situations. The Prisoner's Dilemma was conceived precisely as a tool for investigating this. In the Flood-Dresher experiment [2] done in 1950 a sequence of Prisoner's Dilemma games was conducted by the same pair of players; mutual cooperation was the most common outcome: 60 of the 100 games. Mutual defection, the N.E., occurred only 14 times. Translating this to the ITG, mutual cooperation means $\langle p, p \rangle$ and mutual defection means $\langle q, q \rangle$. If the programmer and the tester behave as in this experiment, their behavior is bad from the lab manager's (and the customer's) point of view.

- If the same game is played repeatedly by the same pair of players, it is not correct from a game-theoretical view to consider these as independent games (as immediately remarked by John Nash in his reaction to the Flood-Dresher experiment, see [2]). The sequence is to be considered as one game in which the two players move pairwise, using the knowledge about the other player's strategy (as built-up so far) in each successive move. In game theory this is known as an 'extensive game with perfect information and simultaneous moves' (see [5] 6.3.2). Now each player has to decide upon a strategy which contains his initial move and also all the answers to all sequences of moves of the other player. Special examples of such strategies are finite state machines. In [5] Sect. 8.4 several such strategy machines are defined. For example one machine M_1 for player 1 works as follows: play C (cooperate) as long as player 2 plays C; play D (defect) for three periods, and then revert back to C, if player 2 plays D when he should play C. So the other player is being "punished" for three periods for playing D and then "forgiven". The machine M_2 of player 2 starts by playing C and continues to do so if the other player chooses D. If the other player chooses C then it switches to D, which it continues to play until the other player again chooses D, when it reverts to playing C. If two players play Prisoner's Dilemma using these finite state machine strategies, the game goes into cycling behavior with a cycle-length of 5. Both $\langle C, C \rangle$ and $\langle D, D \rangle$ occur in the cycle, next to $\langle C, D \rangle$ and $\langle D, C \rangle$. If the programmer and the tester work according to such strategies (and other strategies such as the 'grim' strategy in [5] or the well-known 'tit-for-tat' strategy) then the lab manager is likely to observe cycles; it can be expected that the cycles are irregular because of the random effect of the $\langle p, q \rangle$ and the $\langle q, p \rangle$ action profiles (recall that ITG is based on average payoffs).

- It is possible to consider the higher-level strategic game in which each player has to choose a multi-move strategy once; for example the programmer could choose to always take move D and the tester could behave as defined by machine M_2. The interesting question now is whether there exist some kind of optimal multi-move strategy (some clever machine, perhaps) that is optimal for player 1 and similar for player 2. In [5] Sect. 8.10.3 the following result is given: "if the strategic game G has a unique N.E. payoff profile then for any value of T the action profile chosen after any history in any subgame perfect equilibrium of the T-period repeated game of G is a N.E. of G." Here 'subgame perfect equilibrium' refers to a special type of N.E. for extensive games, taking certain credibility considerations into account (see [5] Sect. 6.2 for the details). If the programmer and the tester have to play ITG in a finite repetition and if they behave rational then this result means that they choose $\langle q, q \rangle$, which is good from the lab manager's viewpoint. However there is evidence that experimental subjects do not behave in a way that is consistent with this result (see the discussion in [5] Sect. 8.2, and also see the Flood-Dresher experiment [2]).

- When considering infinitely repeated games there is the complication that it is not a-priori

clear how the payoffs of infinite sequences are to be defined (just adding them leads to infinite values). There are several alternative definitions: the discounting criterion, the limit of means criterion and the overtaking criterion. The set of equilibria is huge. In [5] Sect. 8.2 Osborne remarks: "the fact that it [the behavior of experimental subjects] is consistent with some subgame perfect equilibrium of the infinitely repeated game is uninteresting since the range of outcomes that are so-consistent is vast." Moreover software developments do not run ad infinitum without bringing in new people or changing the rules.

- It is possible to consider players which choose multi-move strategies (grim, tit-for-tat, etc.) but not just two fixed players; rather an evolving population of players is considered. In [11] three different arrangements are discussed. The first arrangement is a historic sequence of tournaments organized by Axelrod in the later 1970. Colleagues (game theorists) submitted strategies. The simple strategy tit-for-tat won (start with cooperative response and then always repeat the other player's previous move). The second arrangement is a computer simulation of large populations of certain basic types of players equipped with stochastic strategies. Many mutation-selection rounds are performed. As a result, the average payoff in the population can change suddenly. Most of the time, either almost all members of the population cooperate, or almost all defect. The longer the system was allowed to evolve, the greater likelihood "for a cooperative regime to blossom." In heterogeneous populations tit-for-tat is not always superior; it is outperformed by another strategy ('Pavlov', see [12] and [13]). The third arrangement is a population that lives on a two-dimensional grid. Players only interact with eight immediate neighbors. A lone cooperator will be exploited by the surrounding defectors and succumbs. But four cooperators in a block can hold their own. Geometric patterns arise that wander accross the board. In one example population [11] the percentage of cooperators gradually takes a stable value of about 32%. Translating these assumptions to the world of software development we can think of a high-tech area (or perhaps the whole world) with software companies, providing programming services and testing services to a limited set of neighboring companies. For details see [11] and [14].

8 Concluding remarks

The results of our investigations show that if the reward for passing tests for the programmer is high enough and the reward for finding an error for the tester is high enough, there is essentially a Prisoner's Dilemma game hidden in the interaction of a programmer and a tester. The main cause of this seems to be in the assumption that the development laboratory manager cannot measure the quality of the software directly; he only sees PASS or FAIL and therefore he cannot distinguish a bad or lazy programmer in combination with a bad or lazy tester from a good and eager programmer and an a good eager tester (in both cases there is a mixture of PASS and FAIL verdicts on average).

The simplest theoretical solution of the game, the N.E. seems to suggest that mutual defection is the expected outcome. The Prisoner's Dilemma is often seen as "an interesting metaphor for the fundamental biological problem of how cooperative behavior may evolve and be maintained" [14]. In the context of testing the situation is reversed since a testing process is set-up with a deliberate, even desirable, conflict of interest built into it. So the N.E. means that both players choose 'quality' rather that 'poor', which is good from the viewpoint of the software development laboratory and its customers. Looking at sequences of games, the theory does not have much predictive power. Experiments and computer simulations tend to show complex, oscillatory behaviors, mostly showing cooperation. Two-dimensional evolutionary simulations show complex moving patterns.

The complex, oscillating and moving patterns are quite consistent with the image of a software crisis and hard-to-eliminate bugs sketched in [1]. In view of our findings it seems wise for a software development lab manager to make sure that he separates the roles of programmers

and testers and that he makes sure there is enough reward for a programmer to achieve PASSes and for a tester to achieve FAILs. Although our game-theoretical analysis was done using numerical values for the payoff matrix, it is certainly possible that rewards in real life include items such as a feeling of professional responsibility and personal pride.

Disclaimers: it is necessary to relativize the findings because there are certain assumptions behind the models that need not hold in real life situations. First of all, this is the assumption that there is a precise specification given to both players that guarantees that the verdicts PASS and FAIL are not subject of dispute. Although a frequent assumption in research on formal test generation, this need not be true in real development projects. Another assumption is that testing by a professional tester is the only way of determining the quality of the programmer's work. This assumption does not hold if there are other mechanisms at work such as code-walkthroughs, beta-testing, or program correctness proving. Yet another assumption is that neither defect-free software nor full-coverage tests are possible (at or near $Q = 100\%$ and $P = 100\%$ the theory turns into a degenerate case). Finally the ITG and TCTG do not cover the intricacies of real software development: writing a program involves subtle tradeoffs between development time, run-time efficiency, code-size, reusability etc. We constructed another game where finite state machines were tested by finite sequences (programmer effort being automaton size, test effort being test sequence length). We found a Prisoner's Dilemma in this too (details not included in the present article). But, similarly to the text copy testing game this is still an enormous simplification. Programming languages such as C++ and Java are very complex and so are test languages such as TTCN; moreover defining and measuring metrics for programmer's productivity, software quality and test coverage are rich research subjects by themselves.

Acknowledgements

Some of the ideas behind this work described in this article originated in the context of the Côtes de Resyste project and during discussions with Nicu Goga, Sjouke Mauw and Jan Tretmans. The author likes to thank Pieter Cuijpers for his constructive feedback on earlier versions of the paper.

A The text copy testing game

In order to validate the concepts of ITG we develop a more concrete game in which real testing is going on. This game, called TCTG, for Text Copy Testing Game is still not really about software testing, but at least it is about testing. It will be formally analyzed to see whether it is approximated by ITG.

In TCTG The task for the implementer is to transcribe a given text, perhaps from one character set to another, or perhaps even simpler, to type precisely the characters of a given string. The length of the specification and the length of the implementation are the same, L say. The task of the tester is to select one or more positions in the range 1 to L. If the implementation and the specification differ at one or more of the selected positions then an error is found and the verdict is FAIL. Otherwise the verdict is PASS. If the tester chooses several positions, they all have to be different.

As in ITG the implementer and the tester are rational players who choose a performance level first and after that perform a job according to the expectations set by the chosen performance level.

It is convenient to introduce numerical values for the performance levels. In ITG there are two levels but here many levels could exist. The performance level for the implementer is modeled as a variable Q (the Quality of the implementation). The performance level for the tester is modeled as a variable P (the Power of error detection of the test). Both Q and P are in the range from 0 to 1 (the values of 0 and 1 are included). Sometimes they are conveniently expressed as percentages. So instead of choosing a p (poor) performance level the implementer may choose $Q = 25\%$ and instead of a q (quality) performance level the implementer may choose $Q = 50\%$. In the same way the tester may choose $P = 25\%$ or $P = 50\%$, respectively. As in ITG the idea is that equal performance levels lead to a probability of error detection P(FAIL) of about 50%. Indeed, it will turn out that $Q = P = 50\% \Rightarrow$ P(FAIL) $= 50\%$ but because of some of the details of the text copying and testing procedures this idea only holds by approximation for other value pairs for Q and P. The performance level variables are a useful concept but for each value pair of Q and P, the precise probability P(FAIL) has to be calculated as a function $f(Q, P)$. It is natural to demand that the efforts of the implementer and the tester increase monotonically when Q and P increase, respectively. It would be nice for the implementation effort to be a linear function of Q and for the testing effort to be a linear function of P. In the text copy testing game this turns out to hold for P indeed but for Q it only holds as a very rough approximation. This concludes the general introduction of the numerical performance levels; next they must be defined for the specific situation of text copying and text testing.

It is an error if at a specific position the typed character differs from the specified character. In a text of L characters any number of errors E between 0 and L is possible (0 and L inclusive). The implementer's performance level Q is defined by $Q = \frac{1}{E+1}$. For example when $L = 100$, the range of Q is from 0.99% (100 errors) to 100% (no errors). Other definitions are conceivable, for example $\frac{L-E}{L}$ but we consider it inappropriate to assign a relatively high value of 50% to a text in which half of the characters are wrong (in software code it would be ridiculous to even look at software in which 50% of the code statements are wrong). The tester's performance level P is defined by $P = \frac{T}{L}$ where T is the number of positions tested. The range of P is from 0% (nothing tested) to to 100% (everything tested). The advantage of the current definitions is that $Q = 50\%$ and $P = 50\%$ nicely outbalance each other.

Next the relations between the efforts and the performance levels must be defined. These relations depend on the way of working of the implementer and the tester. For the implementer it is assumed that there is a mechanism such that spending more effort leads to a reduction in the number of errors. An example of such a mechanism is a *voting text editor*: each character is typed n times for a given number $n \in \{1, 3, 5, 7, \ldots\}$ etc. and whenever at least $\frac{n+1}{2}$ of these typings are the same, that determines a winning character which goes into the implementation. Otherwise a special error characters is taken. Note that this relation satisfies the monotonicity condition Although the usage of voting mechanisms is not widespread in software engineering, the idea of using redundancy in software design has been proposed under the name "distinct

programming" by Tom Gilb [15].

The implementer has a built-in error rate e, for example $e = 0.1$ which cannot be changed (except by exploiting the voting mechanism). For the tester it is assumed that his effort is proportional to the number of positions tested.

Given these assumptions, the numerical values of Q and P can be determined for various effort values. For $L = 100$ and $e = 0.1$ the expected number of errors with a repetition rate of 1, denoted as $\mathcal{E}(\#\text{ errors} \mid 1\times)$ equals $e \times L = 10$. So $Q = \frac{1}{11} = 9\%$. For triple repetition $\mathcal{E}(\#\text{ errors} \mid 3\times) = (\text{ P(2 errors)} + \text{P(3 errors)}) \times L = (3.(1-e).e^2 + e^3) \times L = (3 \times 0.9 \times 0.01 + 0.001) \times L = 0.028 \times L = 2.8$ whence $Q = \frac{1}{2.8+1} = 26\%$. For five-fold repetition $\mathcal{E}(\#\text{ errors} \mid 5\times) = (\frac{5\times4}{2} \times (0.9)^2 \times 0.001 + 5 \times 0.9 \times 0.0001 + 0.000\,01) \times L = 0.86$ whence $Q = \frac{1}{1.86} = 54\%$. Also $\mathcal{E}(\#\text{ errors} \mid 7\times) = (\frac{7\times6\times5}{3\times2} \times (0.9)^3 \times 0.0001 + \frac{7\times6}{2} \times (0.9)^2 \times 0.000\,01 + 7 \times 0.9 \times 0.000\,001 + 0.000\,000\,1) \times L = 0.27$ whence $Q = \frac{1}{1.27} = 79\%$. Finally $\mathcal{E}(\#\text{ errors} \mid 9\times) = 0.08$ so $Q = \frac{1}{1.08} = 93\%$. The following table summarizes the above findings (for $L = 100$ and $e = 0.1$):

#repetitions	Q
1	9%
3	26%
5	54%
7	79%
9	93%

The relation between the effort (here the number of tested positions) and the power of error detection P (the tester's performance level) is summarized by the following table (for $L = 100$):

#positions	P
10	10%
30	30%
50	50%
70	70%
90	90%

For a game theoretic analysis only the reward/effort ratio is important and therefore the fixation of the absolute effort values is postponed.

B Analysis of the text copy testing game

The first question is how P(FAIL) depends on Q and P as a function $f(Q, P)$ to be determined. First consider a few simple cases for $L = 100$. Let there be one error ($Q = 50\%$) and let there be 50 tested positions ($P = 50\%$). Then P(FAIL) = 50% (the probability that this error is one of these 50 positions out of 100). Let there be 3 errors, $E = 3$, so $Q = \frac{1}{E+1} = \frac{1}{4}$ and let there be 50 tested positions ($P = 50\%$ again). P(FAIL) $\approx 1 -$ (P(given error not found))^3 $= 1 - (1-P)^3 = 1 - (0.5)^3 = 0.875$, that is 87.5%. In general, if $E \ll L$ then P(FAIL) $\approx 1 - (1-P)^E$ and since $E = \frac{1}{Q} - 1$ the following formula for $f(Q, P)$ is adopted:

$$f(Q, P) \;=\; 1 - (1-P)^{\frac{1}{Q}-1}$$

For which combinations of Q and P does P(FAIL) = 50% hold? Easy calculations show $f(0.5, 0.5) = f(0.25, 0.21) = f(0.125, 0.094) = 50\%$. This shows that the iso-P(FAIL) line of 50% does not always run through the points defined by the equation $Q = P$, but for the points shown here it is pretty close.

Next it is time for playing the game. The implementer chooses between $Q = 20\%$ and $Q = 50\%$. After that he chooses randomly among $\binom{100}{4}$ implementations (if $Q = 20\%$) or 100 implementations (if $Q = 50\%$). The tester chooses between $P = 20\%$ and $P = 50\%$. After

that he chooses randomly among $\binom{100}{20}$ tests (if $P = 20\%$) or $\binom{100}{50}$ tests (if $P = 50\%$). The cost of effort difference between $Q = 20\%$ and $Q = 50\%$ is put equal to 1 (think of it as 1\$, perhaps). Interpolating between 1 and 3 repetitions, $Q = 20\%$ occurs at 2.3 repetitions and interpolating between 3 and 5 repetitions, $Q = 50\%$ occurs at 4.7 repetitions; in other words, the $4.7 - 2.3 = 2.4$ extra repetitions (on average), being $2.4 \times L = 240$ key strokes cost 1\$ extra, or 0.42 \$c per key stroke. Similarly the cost of the difference between $P = 20\%$ and $P = 50\%$ is put equal to 1. The difference is 30 positions so each position selected and inspected by the tester costs 3.33\$c. It is convenient to assume for $Q = 20\%$ and $P = 20\%$ the efforts to be equal to 0 (this is convenient; it is irrelevant for the game-theoretic analysis). Let the reward/effort ratio (R) be set to 5.

What is P(FAIL)? It depends on Q and P. For $Q = P = 20\%$, P(FAIL) = 0.59. For $Q = 50\%$, $P = 20\%$ it is found that P(FAIL) $= f(50\%, 20\%) = 1 - (0.8)^{\frac{1}{0.5}-1} = 0.2$ and as calculated before, for $Q = P = 50\%$, P(FAIL) = 50%. So there is a difference with the ITG, where $\langle q, p \rangle$ turned P(FAIL) into 0, whereas here the analogous $\langle Q = 50\%, P = 20\% \rangle$ still leaves a non-neglectable P(FAIL) = 0.2 and similarly $\langle p, q \rangle$ gives P(FAIL) = 1 in ITG but the analogous probability is only $= 1 - (0.5)^{\frac{1}{0.2}-1} = 0.9375$ here. It will be interesting to see whether it still is a Prisoner's Dilemma.

On the basis of the abovementioned assumption the payoff matrix is determined (recall the effort difference of 1 and the reward/effort ratio of 5). Instead of the $\binom{100}{4} \times \binom{100}{20}$ entries formally required in the $Q = 20\%$, $P = 20\%$ quadrant of the payoff matrix, a more convenient notation is possible: only one entry is necessary to calculate the average payoffs (and similarly for the other three quadrants). This entry contains the payoff values for PASS and the payoff values for FAIL (in the given order), separated by a slash. Using this representation the payoff matrix is:

	P=20%	P=50%
Q=20%	5,0/0,5	5,−1/0,4
Q=50%	4,0/−1,5	4,−1/−1,4

The same representation can be used for the weighting matrix containing the probabilities of PASS and FAIL. The weighting matrix is:

	P=20%	P=50%
Q=20%	0.41/0.59	0.0625/0.9375
Q=50%	0.80/0.20	0.50/0.50

The payoff matrix and the weighting matrix can be multiplied in an element-wise fashion to get:

	P=20%	P=50%
Q=20%	2.05,0/0,2.95	0.3125,−0.0625/0,3.75
Q=50%	3.2,0/−0.2,1	2,−0.5/−0.5,2

And by pair-wise adding the payoffs for PASS and FAIL the following payoff matrix is obtained (after rounding off 0.3125 to 0.31 and 3.6875 to 3.69). Call this abstract 2 × 2 game TCTG:

	P=20%	P=50%
Q=20%	2.05,2.95	0.31,3.69
Q=50%	3,1	1.5,1.5

Please observe that this TCTG is essentially a Prisoner's Dilemma (as characterized by the fact that it has one N.E. which is not optimal for both players). It is not as nice and symmetric as the ITG; the differences come mostly from the fact that P(FAIL) takes irregular values, not precisely 0%, 50% or 100%.

References

[1] W.W. Gibbs. Software's chronic crisis. Scientific American, Sept. 1994, pp. 72–81 (1994).

[2] W. Poundstone. Prisoner's dilemma, Doubleday ISBN 0385-41567-2 (1992).

[3] J. Von Neumann. Zur Theorie der Gesellschaftsspiele, Mathematische Annalen, 100, pp. 295–320 (1928).

[4] J.F. Nash. Equilibrium points in N-person games, Proceedings of NAS (1950).

[5] M.J. Osborne, A. Rubinstein. A course in game theory, MIT Press (1994).

[6] E. Freeman. Building Gargantuan Software, Scientific American Presents, Vol. 10, N. 4, Special issue on extreme engineering, pp. 28–31 (1999).

[7] OSI. Conformance testing methodology and framework, Part 3: The Tree and Tabular Combined Notation (TTCN), ISO/IEC DIS 9646-3 (1990).

[8] S. Vuong, W. Chan, M. Ito. The OIUv method for protocol test sequence generation, In: Second International Workshop on Protocol Test Systems, Berlin, Oct. (1989).

[9] J.R. Moonen, J.M.T. Romijn, O. Sies, J.G. Springintveld, L.M.G. Feijs, R.L.C. Koymans. A two-level approach to automated conformance testing of VHDL systems, In: M. Kim, S. Kang, K. Hong (Eds.), IFIP TC6 International Workshop on Testing of Communicating Systems, Chapman & Hall, pp. 432–447 (1997).

[10] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence, Software – Concepts and Tools, 117:103–120 (1996).

[11] M.A. Nowak, R.M. May, K. Sigmund. The arithmetics of mutual help, Scientific American, June 1995, pp. 50–53 (1995).

[12] M.A. Nowak, K. Sigmund. Tit for tat in heterogeneous populations, Nature, Vol. 355, pp. 250–253 (1992).

[13] M.A. Nowak, K. Sigmund. A strategy of win-stay lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game. Nature, Vol. 364, pp. 56–58 (1993).

[14] M.A. Nowak, R.M. May. Evolutionary games and spatial chaos. Nature, Vol. 359, pp. 826–829 (1993).

[15] T. Gilb. Distinct software: a redundancy technology for reliable software, in: Infotech State of the Art Report on Software Reliability, Pergamon Infotech, Maidenhead, UK. (1977).