Olympiads in Informatics, 2011, Vol. 5, 32–43
© 2011 *Vilnius University*

Codility. Application of Olympiad-Style Code Assessment to Pre-Hire Screening of Programmers

Grzegorz JAKACKI[1], Marcin KUBICA[2], Tomasz WALEŃ[1,2]
[1] *Codility Ltd., London, UK*
[2] *Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland*
e-mail: {jakacki,walen}@codility.com, {kubica,walen}@mimuw.edu.pl

Abstract. Over past 20 years programming contests have grown substantial expertise in code assessment, yet it is rarely applied outside the contest settings. This paper presents a case of successful adoption of Olympiad-style code assessment technology in Codility, a commercial web service intended for pre-hire screening of software developers. We discuss correctness and soundness of the automated pre-hire screening, and characterise the class of programming tasks best suited for such application.

Key words: programmers recruitment, pre-hire screening, program evaluation, time complexity assessment, model-checking.

History

In 2005 one of the authors joined Exoweb.net, an outsourced software development business in Beijing, China. At that time Exoweb, a company with a workforce not exceeding 10 software developers, managed to secure a new software development contract generating an immediate demand for another 10 software developers (a demand which only continued to grow for many subsequent months). It has to be noted that due to the character of the contract Exoweb was very cautious about the overall skill levels of prospective employees. Based on previous experiences and the existing literature on the subject (most notably Spolsky, 2000; Weinberg, 1998), Exoweb management designed a recruitment process in which each candidate had to undergo three successful independent interviews with senior software engineers. Moreover, Exoweb management decided that adherence to item 11 of so-called *Joel Test* was crucial, i.e., that each candidate had to convincingly demonstrate the ability to write correct code in some imperative language (the choice of the language was left to the candidate).

As soon as Exoweb started advertising the job positions, it became apparent that the interviews with senior engineers were the main bottleneck of the recruitment process. There were only three software engineers in the company able to conduct the interviews. The hires-to-candidates ratio was around $1 : 50$ and the time spent in interviews with one candidate was 20–180 min with an average of approximately 100 min, yielding the approximate amount of 80 hours of senior engineer's time per one hire.

The interviewers observed that an easily identifiable reason of many failed interviews was the inability to write bug-free code. In an attempt to reduce the number of interviews, Exoweb management decided to administer a written programming to each candidate prior to interviews. Soon the test has been moved from paper to computer and augmented with an automated checker (Exobench) developed in-house in the spirit of checkers employed by IOI and ACM CPC. Exobench was implemented as a command-line tool and non-technical admin staff has been trained to operate it without assistance of engineering staff.

It turned out that approximately 9 in 10 candidates were being reliably rejected without wasting any senior engineer's time in the interviews. Moreover, the admin staff able to operate the screening software was much cheaper (with salaries lower by a factor of 3–5) and much easier to hire than technical interviewers. The total amount of senior engineer's work per one hire dropped from 80 hours to 12 hours. The process has been subsequently executed for 30 months without major modifications, covering 2.5 thousands applicants, with automated pre-screening efficiency of 1 : 10 and the interview rejection rate of 1 : 5.

In 2008, given the successful validation at Exoweb, the idea of automated pre-hire screening has been re-implemented as an on-line service and offered under the trade name Codility.

It is worth mentioning, that the founders of Codility are all former IOI contestants and medalists, and they are (currently or have been previously) associated with Polish Olympiad in Informatics (POI). The two sources of experience, the industry and the Olympiad (Diks *et al.*, 2007), have been augmented to create this innovative and successful enterprise.

Tasks and Task Preparation

The purpose of pre-hire screening differs much from the goals of programming contests. In both cases the focus is on differentiating participants according to their skills. In case of programming contests the goal is to select top contestants and fairly distribute points among all the other contestants. On the other hand, the goal of pre-hire screening is to eliminate all the applicants not possessing sufficient programming skills. Hence, the tasks used at Codility are generally much more elementary (cf. Verhoeff *et al.*, 2006; Diks *et al.*, 2008). Also, the assessment result is more of a binary nature (reject or send for interviews), than in IOI. It is quite natural, since the final outcome is also binary – the goal is to select all the applicants that can be of interest for the employee. It is not important to differentiate those that are eliminated, and those who pass will undergo further selection procedures anyway.

The variety of programming languages, that can be used by applicants is much wider than in IOI or other programming contests. The reason is, that applicants can hardly be expected to train focusing on Codility settings. To the contrary, employees expect Codility to asses applicants' programming skills, regardless of the particular programming languages they are using. Hence, Codility system supports most popular programming languages, including: C, C++, C#, Java, JavaScript, Pascal, Python, Perl, PHP and Ruby.

That poses challenges on both task preparation and evaluation. The latter ones include comparable assessment of solutions written in fully compilable languages (such as C and C++), byte-code level compilable languages (such as Java and Python) and scripting languages (such as PHP).

Task preparation consists of similar elements as in programing contests (cf. Diks *et al.*, 2008): task formulation, model solutions, alternative slower solutions, example wrong solutions tests and result checker. Some details however, are different.

Typical task formulation is much shorter and simpler than in IOI. Narrative story is not present. Textual problem description is usually accompanied by an example, input data constraints (sometimes implicit), formal problem definition and interface specification. The input-output interface slightly differs from interfaces used at IOI or other Olympiads in Informatics. The task solution comprises a function written to a specific interface, which significantly reduces the burden of parsing/unparsing the data – the input data is given as function arguments, the output data is passed via return value. In IOI terminology, the tasks are of batch nature. Interactive and open-data tasks are not supported, and actually – not needed. For each supported programming language, an interface specification and the formulation are automatically augmented to form the whole task statement.

As we have already mentioned, tasks used by Codility are much easier, than those that can be seen at programming contests. Still, similarly to IOI, there usually exists a trivial, but suboptimal solution. The ability to produce such a solution proves only elementary programming skills, while producing the optimal solution requires some algorithmic skills. An example task formulation can be found in Appendix A.

The number of supported programming languages is significant, and what more important, evolves all the time. Developing all required solutions in all the programming languages would be a burden. Moreover, the number of tasks that are ready to be used and are maintained exceeds a hundred and is growing, and extending tasks analyses whenever a support for a new programming language is included would be simply unfeasible. Hence, a more generic and flexible framework had to be developed. Python has been chosen as a "front-end" programming language – solutions written in (a carefully chosen subset of) Python are automatically translated to all the other supported programming languages. Contrary to IOI practice, the test cases are not pre-determined and are replaced by a testing module (written in Python) which (among other checking procedures) runs solution against dynamically generated test cases. With a support of dedicated libraries encapsulating particular execution environments, programs written in any supported programming language can be run against the Python tests in an uniform manner. Time limits are automatically calibrated for all the supported languages independently, based on running times of model solutions automatically translated into target languages.

Tests are prepared according to similar outlines, as in Olympiads in Informatics (Diks *et al.*, 2008). They include:

- correctness tests of various sizes,
- border-cases tests,
- efficiency tests.

Compared to IOI and other Olympiads in Informatics, there is more stress on correctness and border-cases, and time complexity is a bit less important. Due to the fact that test-case execution is not a pre-determined set of test-runs, but rather a function in Python test module, arbitrary evaluation logic can be easily implemented, including execution of multiple test-runs under one test-case to emulate IOI test grouping.

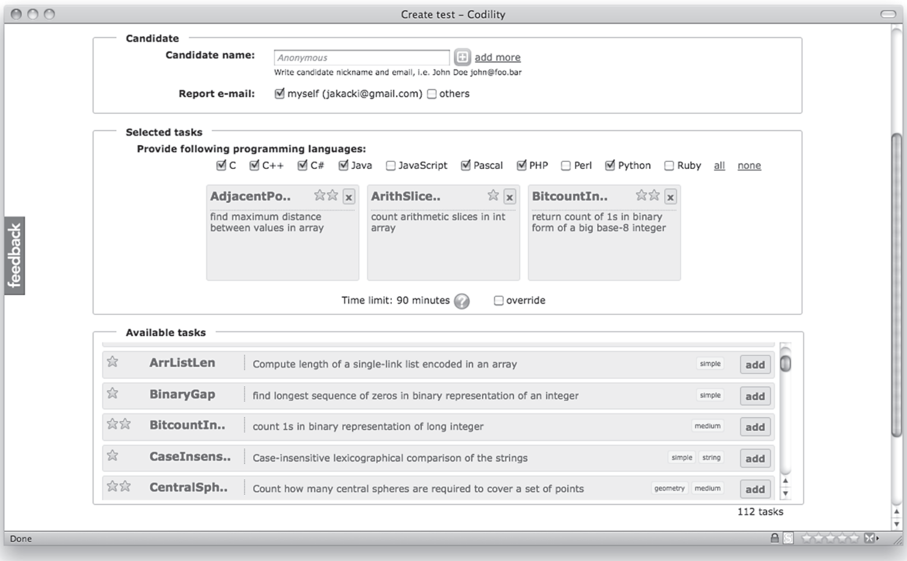
Checker as a Web Service

Codility operates as a website (codility.com) offering pre-screening service to software recruiters. As such, the service has two main user types: candidates and recruiters. Generally users of each type interact with the system through a separate interface.

From the very beginning Exobench (and later Codility) were designed to minimize the need for technical staff supervision in the very first line of screening of candidates. Contrary to IOI, no technical supervision was intended during the test and tests were meant to be applied by admin staff without extensive technical training. Furthermore, the nature of the recruitment process called for tests that are less "ruthless" than in IOI, in particular some tolerance to simple, avoidable mistakes (e.g., wrong name of the function, missing return statement) was desired. These factors led to introduction of the candidate's interface. Initially, it was a CLI command, enabling the candidate to compile the solution and run it against a small pre-defined subset of test-cases (usually just the example test-case discussed in the problem formulation). Ability to compile the code and execute a simple test with one command gave candidates a quick way to run a sanity check, at the same time ensuring that they have a chance to fix simple mistakes before submitting the solution. In such a setup it was also easy to assure that the execution of a solution via candidate's and recruiter's interfaces take place in the same controlled environment, minimizing the discrepancies due to different compiler options, different ways of calling the solution function, different seeds of pseudo-random generators etc. Candidate's interface bears some similarity to the contestant's interface present in ACM CPC. This contest enables contestants to submit solutions for evaluation during the competition as well as obtain the information about correctness of the submission. The main difference is that Exobench/Codility by design limits the number of test-cases executed as a part of sanity check to reduce the turn-around time and to prevent the technique (well-known among CPC contestants) of extracting information about test-cases through multiple submissions. Contrary to ACM CPC, Codility does not penalize candidates for multiple sanity check submissions.

When the system has been re-implemented as web service, the candidate's interface took form of a micro-IDE consisting of:

- task description pane,
- code editor,
- control buttons, including the "VERIFY" button, enabling the candidate to run the compilation and sanity check,
- countdown timer.



Candidate

Candidate name: Anonymous add more

Write candidate nickname and email, i.e. John Doe john@foo.bar

Report e-mail: [✓ myself (jakacki@gmail.com)] others

Selected tasks

Provide following programming languages:

[✓ C] [✓ C++] [✓ C#] [✓ Java] JavaScript [✓ Pascal] [✓ PHP] Perl [✓ Python] Ruby all none

AdjacentPo..	ArithSlice..	BitcountIn..
find maximum distance between values in array	count arithmetic slices in int array	return count of 1s in binary form of a big base-8 integer

Time limit: 90 minutes override

Available tasks

☆	ArrListLen	Compute length of a single-link list encoded in an array	simple	add
☆	BinaryGap	find longest sequence of zeros in binary representation of an integer	simple	add
☆☆	BitcountIn..	count 1s in binary representation of long integer	medium	add
☆	CaseInsens..	Case-insensitive lexicographical comparison of the strings	simple string	add
☆☆	CentralSph..	Count how many central spheres are required to cover a set of points	geometry medium	add

112 tasks

Done

Fig. 1. Codility recruiter’s interface for building new tests.

Subsequently more features were added, in particular ability to add candidate’s own tests to the sanity check run.

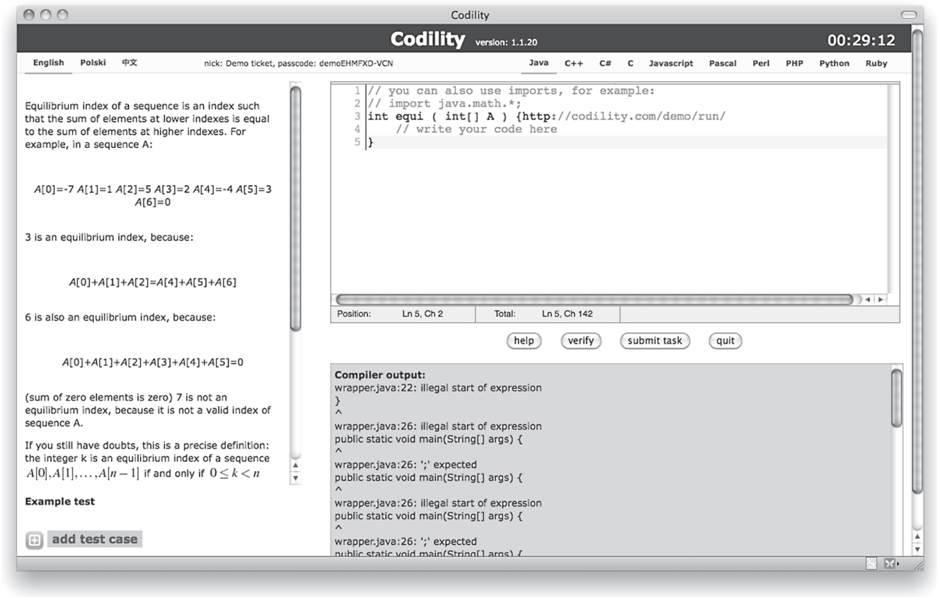
The recruiter’s interface of Codility contains several screens and is mostly focused on building, replicating and administering the tests, as well as reviewing the candidate’s ranking and detailed results of tests.

Evaluation Technology

Codility was initially based on the evaluation model evolved by IOI and ACM CPC, i.e., execution of test-runs in the sandboxed environment. Codility augmented this model with several technologies, two of which are discussed below.

Time Complexity Assessment

Codility test report consists of a numeric score, measuring the quality of the solution (much like the score in IOI) together with a detailed report, including the test results for each test-run. The test-run results constitute documentation of solution shortcomings and provide additional insight for a technical recruiter. Codility team has observed however, that compared to IOI, the software engineer looking at the test-run results has a limited knowledge about the nature of the test-runs. Moreover, it has been identified, that a single most important piece of information that the recruiting engineer is trying to infer from the test-run results is the assessment of the performance scalability of the solution. This observation led to the development of proprietary technology for assessment of asymptotic



nick: Demo ticket, passcode: demoEHMFXD-VCN

Equilibrium index of a sequence is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes. For example, in a sequence A:

A[0]=-7 *A*[1]=1 *A*[2]=5 *A*[3]=2 *A*[4]=-4 *A*[5]=3 *A*[6]=0

3 is an equilibrium index, because:

A[0]+*A*[1]+*A*[2]=*A*[4]+*A*[5]+*A*[6]

6 is also an equilibrium index, because:

A[0]+*A*[1]+*A*[2]+*A*[3]+*A*[4]+*A*[5]=0

(sum of zero elements is zero) 7 is not an equilibrium index, because it is not a valid index of sequence A.

If you still have doubts, this is a precise definition: the integer k is an equilibrium index of a sequence $A[0], A[1], \ldots, A[n-1]$ if and only if $0 \le k < n$

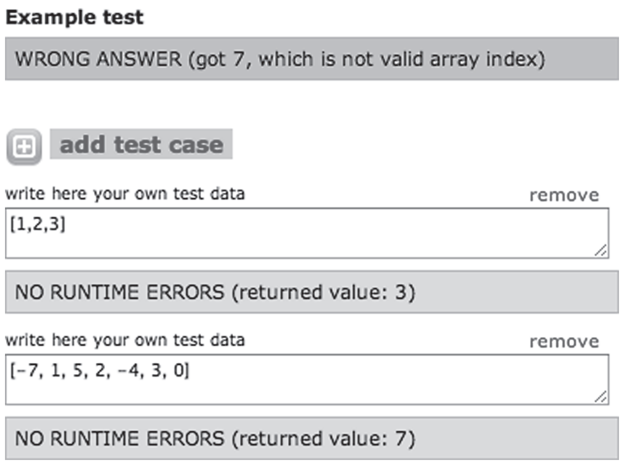
Example test

```
// you can also use imports, for example:
// import java.math.*;
int equi ( int[] A ) {http://codility.com/demo/run/
    // write your code here
}
```

Position: Ln 5, Ch 2 Total: Ln 5, Ch 142

```
Compiler output:
wrapper.java:22: illegal start of expression
}
^
wrapper.java:26: illegal start of expression
public static void main(String[] args) {
^
wrapper.java:26: ';' expected
public static void main(String[] args) {
^
wrapper.java:26: illegal start of expression
public static void main(String[] args) {
^
wrapper.java:26: ';' expected
public static void main(String[] args) {
```

Fig. 2. Candidate's interface (micro-IDE). Task description shown in the left pane; code editor and error console in the right panes.



Example test

```
WRONG ANSWER (got 7, which is not valid array index)
```

write here your own test data

```
[1,2,3]
```

```
NO RUNTIME ERRORS (returned value: 3)
```

write here your own test data

```
[-7, 1, 5, 2, -4, 3, 0]
```

```
NO RUNTIME ERRORS (returned value: 7)
```

Fig. 3. Part of web interface for adding candidate's own testcases.

computational complexity of the solutions. The assessment engine attempts to derive a simple closed form of the function bounding the running time. The results are presented as an expression in O-notation.

Model Checking

Test-run based evaluation is definitely a proven way of assessing the functional correctness of solutions, however in general the equivalence of the model solution and the tested solution cannot be established computationally (which is rather fundamental corollary of Rice's theorem). Any finite set of test-runs is doomed to contain "holes". Imperfections of a particular solution may manifest themselves just in these uncovered holes, remaining undiscovered by the particular test-run set. Codility has conducted a research to find out how often solutions are "overscored" due to insufficient test-run coverage. The discussion of this research is beyond the scope of this paper, however it led to another interesting augmentation of checking technology. Codility checker employs a hybrid of random test generation, enumerative test generation, symbolic interpretation and model checking to proactively "cover the holes", i.e., to generate the test-runs aimed at derailing particular overscored solutions.

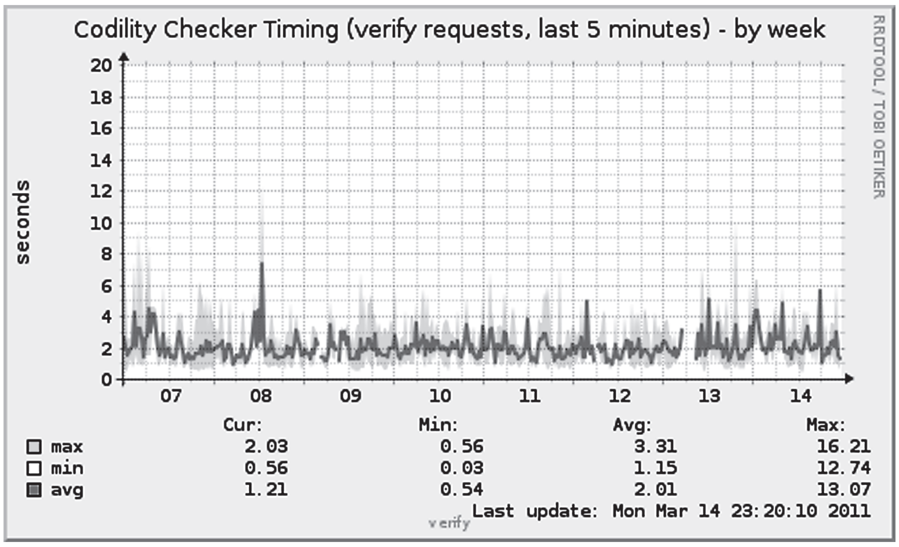
Scalability

Unlike IOI, part of Codility checking has to happen in real time. Candidates' interface enables execution of sanity checks, including compilation and execution of certain test-runs. To be of any use, the feedback from these checks has to be presented to candidates quickly, ideally within seconds. This requirement calls for an architecture which can provide enough computing power to run the checkers in case of demand spikes. Codility is implemented as highly scalable distributed application within Amazon EC2 computing cloud. Elastic architecture enables on-the-fly addition of computing power if the performance metrics drop below threshold levels. Beyond standard performance metrics (e.g., load average) carefully chosen set of application-specific metrics is constantly monitored, including (a) the number of queued checker jobs, (b) min/avg/max checking time within last 5 minutes, (c) number of ongoing tests. To assure service quality for customers in diverse geographies, Codility servers are distributed over data-centers in US, Ireland and Singapore.

Correctness and Soundness

Risking a gross oversimplification we may say that evaluation systems, like IOI checkers or Codility, attempt to assess a group consisting of good programmers and poor programmers. The goal is to obtain a concrete division of the group into winners and losers, so that the set of winners most closely matches the set of good programmers, and the set of losers most closely matches the set of poor programmers. The notion of being good or poor programmer is intuitive rather than well-defined and the difficulty of building an evaluation systems lies in approximating these fuzzy concept with concrete divisions into winners and losers.

Codility team has identified two desired properties of an evaluation system (named after similar properties of formal systems in mathematical logic):



```
                Cur:          Min:          Avg:          Max:
□ max           2.03          0.56          3.31         16.21
□ min           0.56          0.03          1.15         12.74
■ avg           1.21          0.54          2.01         13.07
                         Last update:  Mon Mar 14 23:20:10 2011
```

Fig. 4. Min/avg/max turnaround times of "VERIFY" checks tracked by Munin on Codility production servers.

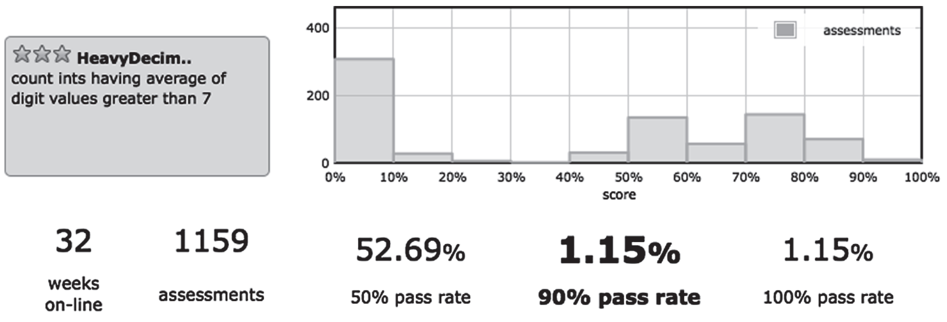
- **correctness** – all individuals evaluated as winners are good programmers (all bad programmers are evaluated as losers),
- **soundness** – all good programmers are evaluated as winners (all individuals evaluated as losers are poor programmers).

Furthermore, Codility team has observed that different properties are more desired in a contest settings and in a pre-hire screening settings.

Correctness is crucial for contests. It would be detrimental to the contest reputation if the winners are proven to be inferior programmers. Correctness is also critical to reaching the goal of regional contests, aiming to maximize the chances of the regional representatives. Soundness, on the other hand, is good to have, but less crucial. It is an accepted fact, that a programming champion may not succeed in a particular competition due to bad luck, bad day, etc. Undoubtedly evaluation system which can provide soundness is desired, but occasional lack of soundness is acceptable.

In pre-hire screening correctness is not considered crucial. Due to the nature of the pre-screening process (high-pass filter before more elaborate in-person interviews), it is accepted that occasionally a poor programmer sneaks through the evaluation system, due to sheer luck, pre-existing knowledge of test problems or cheating (e.g., impersonation by a smarter colleague in an on-line test). As long as such incidents are infrequent, the screening efficiency is not seriously affected. Codility surveyed professional recruiters about their perceived percentage of candidates who cheat in on-line pre-screening tests. The answers varied, but never exceeded 10%. With Codility screening efficiency reaching 90%, even the most pessimistic scenario yields a screening ratio exceeding 80%.

Soundness, on the other hand, seems very important in pre-hire screening. Programming contests see abundance of talent and generally there is always enough candidates to put somebody on the podium, but in recruitment the talent is scarce. In organizations that



☆☆☆ **HeavyDecim..**
count ints having average of digit values greater than 7

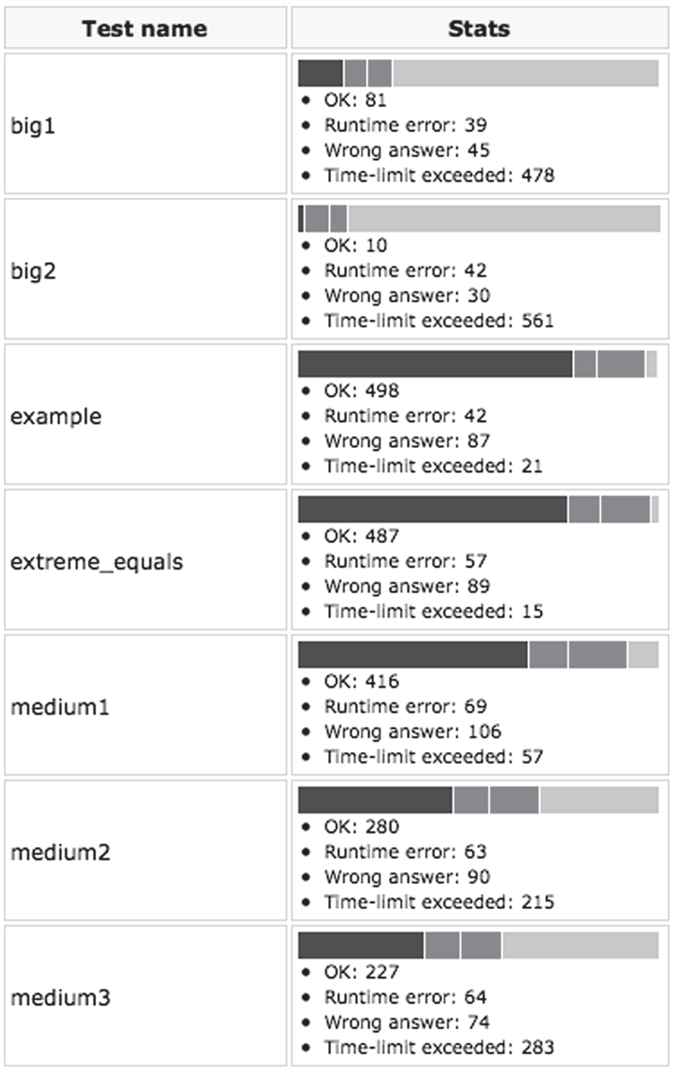
32	1159	52.69%	**1.15%**	1.15%
weeks on-line	assessments	50% pass rate	**90% pass rate**	100% pass rate

Fig. 5. Fragment of Codility webpage showing distributions of scores for a particular problem. Two peaks are visible around popular suboptimal solutions.

have exhausted peer recommendations, recruiters sift through candidates from the job market, most of whom are unfit for the job. At Exoweb, a case study described earlier in this paper, roughly 50 candidates needed to be pre-screened and subsequently 5 of them interviewed for one of them to be hired. With these numbers, a campaign aiming to hire 10 persons needs to pre-screen 500 and interview 50. An evaluation system that evaluates one additional poor programmer as winner per each ten winners, necessitates 5 more interviews. On the other hand, a system that evaluates one good programmer as a looser per each ten good programmers, necessitates over 50 more candidates to be pre-screened. As long as the automated pre-screening is cheap, time occurs to be a crucial factor here – 5 interviews with poor programmers will take up to 5 hours of senior engineers' time, but generating 50 new reasonably matching leads from a job ads in 5 hours is expensive and hardly attainable. (Certain on-line channels in very large job markets, like China, are capable of generating large amounts of leads in short time, however the prospective gain is usually wrecked by drastically high pre-screening rejection rate.) In the recruitment setting soundness is more desired than correctness.

Scoring

Codility scoring system generally does not differ from the one used by IOI. Candidates solve between 1 to 6 problems (with average of 2.24 problems per person), the total score is the sum of scores obtained for individual problems. Contrary to IOI, the problems are written and timed with assumption, that a competent programmer should be able to obtain the score close to 100%. Generally score distributions differ substantially from problem to problem, with numbers of candidates scoring 100% ranging from as high as 48% to as low as 1.15%. 6.9% of all paid Codility evaluations achieved top score, 11.7% achieved score exceeding 90%. Distributions are generally biased towards lower end with peaks around scores corresponding to suboptimal solutions.



Test name	Stats
big1	• OK: 81 • Runtime error: 39 • Wrong answer: 45 • Time-limit exceeded: 478
big2	• OK: 10 • Runtime error: 42 • Wrong answer: 30 • Time-limit exceeded: 561
example	• OK: 498 • Runtime error: 42 • Wrong answer: 87 • Time-limit exceeded: 21
extreme_equals	• OK: 487 • Runtime error: 57 • Wrong answer: 89 • Time-limit exceeded: 15
medium1	• OK: 416 • Runtime error: 69 • Wrong answer: 106 • Time-limit exceeded: 57
medium2	• OK: 280 • Runtime error: 63 • Wrong answer: 90 • Time-limit exceeded: 215
medium3	• OK: 227 • Runtime error: 64 • Wrong answer: 74 • Time-limit exceeded: 283

Fig. 6. Part of Codility analysis of failure reasons for particular test-cases.

Conclusions

Program evaluation methods, developed by IOI and other programming contests, can be successfully applied in pre-hire screening. Codility founders have validated this claim, first in an enclosed setting of one large hiring campaign, later in the market. Today Codility is a profitable business catering to customers like Nokia, Siemens or Barnes&Noble. Codility-funded research led to development of two innovative technologies supporting code evaluation, namely automated time complexity assessment and reactive tests augmentation. Hopefully the innovation in this space will lead to further advances in evaluation technology employed by programming competitions.

References

Diks, K., Kubica, M., Stencel, K. (2007). Polish olympiad in informatics – 14 years of experience. *Olympiads in Informatics*, 1, 50–56.

Diks, K., Kubica, M., Radoszewski, J., Stencel, K. (2008). *A Proposal for a task preparation process*. *Olympiads in Informatics*, 2, 64–75.

Spolsky, J. (2000). *The Guerilla Guide to Interviewing*, Joel Spolsky's blog 2000. `http://www.joelonsoftware.com/articles/fog0000000073.html`.

The Joel Test. `http://www.joelonsoftware.com/articles/fog0000000043.html`

Verhoeff, T., Horváth, G. , Diks, K., Cormack, G. (2006). A proposal for an IOI syllabus. *Teaching Mathematics and Computer Science*, 9(1).

Weinberg, G.M. (1998). *The Psychology of Computer Programming*, Dorset House.

Appendix A

Example Task Description

Equilibrium index of a sequence is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes. For example, in a sequence A:

$$A[0] = -7,\ A[1] = 1,\ A[2] = 5,\ A[3] = 2,\ A[4] = -4,\ A[5] = 3,\ A[6] = 0.$$

3 is an equilibrium index, because:

$$A[0] + A[1] + A[2] = A[4] + A[5] + A[6].$$

6 is also an equilibrium index, because:

$$A[0] + A[1] + A[2] + A[3] + A[4] + A[5] = 0.$$

(sum of zero elements is zero) 7 is not an equilibrium index, because it is not a valid index of sequence A.

If you still have doubts, this is a precise definition: the integer k is an equilibrium index of a sequence $A[0], A[1], \ldots, A[n-1]$, if and only if, $0 \leqslant k < n$ and:

$$\sum_{i=0}^{k-1} A[i] = \sum_{i=k+1}^{n-1} A[i].$$

Assume the sum of zero elements is equal zero. Write a function:

```
int equi(int[] A);
```

that given a sequence, returns its equilibrium index (any) or -1 if no equilibrium indexes exist. Assume that the sequence may be very long.

G. Jakacki (1975), M.Sc. in computer science, CTO&CEO at Codility Ltd., part-time lecturer at Warsaw University, formerly software developer and technical team leader at Exoweb.net and Synopsys Inc., contributor to Polish Olympiad in Informatics. His interests focus on building and motivating software development teams, practical applications of software theory, semantics of programming languages.

M. Kubica (1971), PhD in computer science, assistant professor at Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, scientific secretary of Polish Olympiad in Informatics, CEOI'2010 chairman and former chairman of Scientific Committees of IOI'2005 in Nowy Sącz, CEOI'2004 in Rzeszów, BOI'2001 in Sopot, and BOI'2008 in Gdynia, Poland. His research interests focus on combinatorial and text algorithms.

T. Waleń (1978) PhD in computer science, principal developer and founder of Codility, is a software practitioner and theoretician, lecturer at University of Warsaw. Designed and implemented evaluation solutions used at major programming contests worldwide and was one of principal contributors in USOS, the nationwide education management system. Involved in teaching advanced subjects in computer science, as well as a research on combinatorial algorithms.