World Academy of Science, Engineering and Technology
International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:9, No:12, 2015

Manual to Automated Testing: An Effort-Based Approach for Determining the Priority of Software Test Automation

Peter Sabev, Katalina Grigorova

Abstract—Test automation allows performing difficult and time consuming manual software testing tasks efficiently, quickly and repeatedly. However, development and maintenance of automated tests is expensive, so it needs a proper prioritization what to automate first. This paper describes a simple yet efficient approach for such prioritization of test cases based on the effort needed for both manual execution and software test automation. The suggested approach is very flexible because it allows working with a variety of assessment methods, and adding or removing new candidates at any time. The theoretical ideas presented in this article have been successfully applied in real world situations in several software companies by the authors and their colleagues including testing of real estate websites, cryptographic and authentication solutions, OSGi-based middleware framework that has been applied in various systems for smart homes, connected cars, production plants, sensors, home appliances, car head units and engine control units (ECU), vending machines, medical devices, industry equipment and other devices that either contain or are connected to an embedded service gateway.

Keywords—Automated Testing, Manual Testing, Test Automation, Software testing, Test Prioritization.

I. Introduction

TEST automation can provide a tremendous boost to most teams and organizations in regard of driving testing cycle times down and test coverage up. Furthermore, automated tests produce much faster results, perform precisely the same operations each time they run (thereby eliminating human error), and it is easy to test how the software reacts under repeated execution of the same operations.

However, one has neither the infinite resources, nor the time to automate everything. If one tries to “automate everything”, and picks a set of “low-hanging fruit” automation candidates just to initiate the automatic testing, one will gain a false sense of security that the tool will take care of all the details surrounding automation or will dive into automation development without a clear strategy which test cases to automate, and therefore the automation can easily turn into disaster [8].

Reality shows that automating everything is rarely possible. A research from 2002 found that on average 60% of the overall project tests are automated [6]. However, software has become much more complex in recent years, and this has a negative effect on the automated tests percentage. Industry surveys from 2010 indicated that 75% of all functional testing is still performed manually [3]. But even if it was possible to automate 100% of the tests, this would not happen at once. Development and maintenance of automated tests is 3 to 15 times more expensive compared to manual tests [6], so organizations and teams who want to take advantage of what automation has to offer, need to be cost-effective and time-efficient. Thus, they would need proper prioritization which test cases to automate first.

Peter Sabev and Prof. Katalina Grigorova are with the Department of Informatics and Information Technologies, Faculty of Natural Sciences and Education, “Angel Kanchev” University of Ruse (e-mail: psabev@gmail.com, kgrigorova@uni-ruse.bg).

II. Scope

A number of different approaches have been studied to aid the manual regression testing process. The three major branches include test suite minimization, test case selection and test case prioritization. Test suite minimization is a process that seeks to identify and then eliminate the obsolete or redundant test cases from the test suite. Test case selection deals with the problem of selecting a subset of test cases that will be used to test the changed parts of the software. Finally, test case prioritization concerns the identification of the "ideal" ordering of test cases that maximizes desirable properties [4]. This paper focuses entirely on the test case prioritization for minimizing the manual test effort while quickly achieving a good level of automation, respecting business priorities, cost-effectiveness and early fault detection.

Before considering the prioritization itself, one should know that there are tests which can be executed either manually or automatically. For example, load testing often requires creating heavy user workloads. Even if it was possible to arrange for hundreds of manual testers to test simultaneously, this would have been surely impractical and not cost-effective. Load and performance tests need to be automated as there is no viable manual alternative [7]. Another example when manual execution is impossible is using specific API or other “hidden” properties of the software, which are not available to the end user. On the other hand, there are tests that non-human testing could easily miss. Manual testing may be the only option when specific project features must be validated subjectively by humans such as usability, user experience or look-and-feel, as well as tests with unpredictable (for a machine) results. Creating automated scripts may be a waste of time for new application functions that are still in development; and which are evolving or changing frequently [3]. There are many other factors that could leave one without the option to choose between manual and automated test cases. The list begins with human resources and hardware costs,

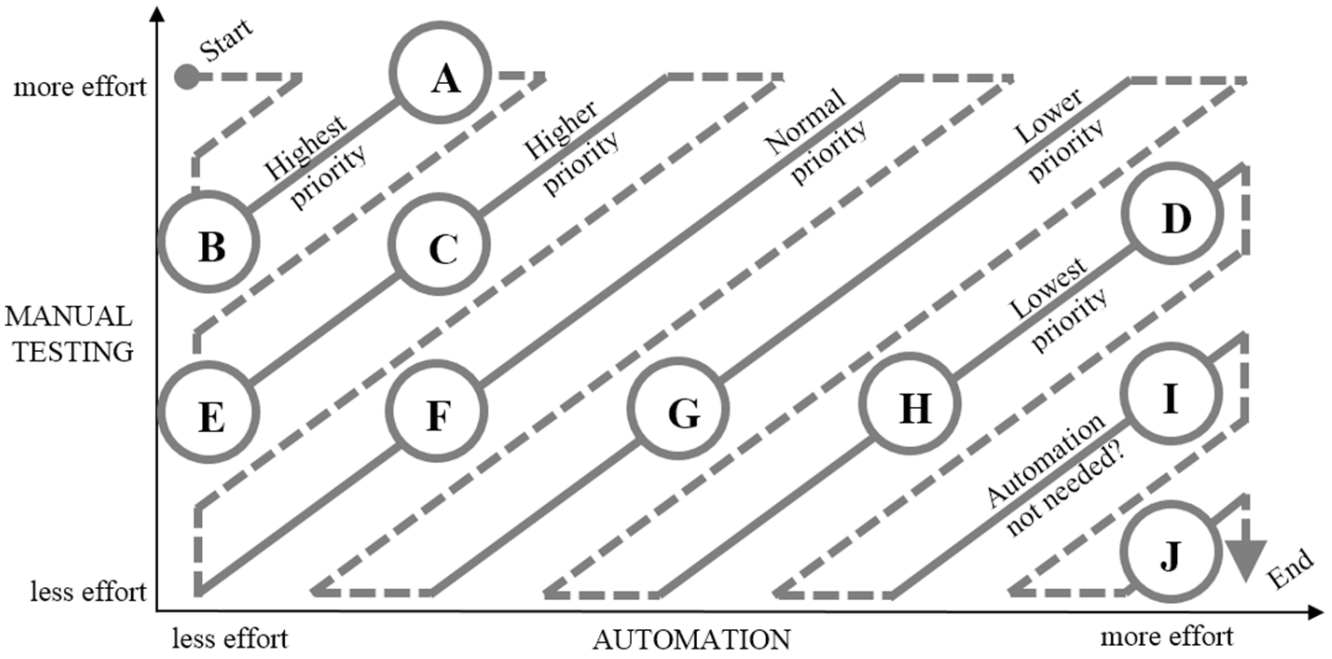

Fig. 1 Prioritization of manual test case candidates suitable for automation

preliminary setup of the test environment, dependency between the test cases, company processes and standards, project schedules, continuous integration and delivery restrictions, law and standard restrictions, etc. However, in the vast majority of projects there will be a set of manual tests that are possible candidates for test automation. These could be executed by both human and automation agents with equal or at least comparable effectiveness. The approach proposed in this paper focuses exactly on those candidates and on the way to prioritize them in order to build effective test automation with good return of investment (ROI).

III. Approach Overview

Fig. 1 illustrates the main idea of the approach. It represents a Cartesian coordinate system where the vertical axis shows the manual testing effort and the horizontal axis shows the effort to automate the test. All test case candidates should be assessed and put as points in that coordinate system. As the figure shows, prioritization starts from the top left (highest priority) and ends at the bottom right (lowest priority). In the example, the order of automation should be B-A-E-C-F-G-H-D-I-J. Note that A and B could be considered as having the same priority, as they lay on the same diagonal. This is the same for C and E, and for D and H. If the effort estimations are correct, this approach will produce maximum ROI, because it will save lot of manual testing effort by exercising minimal automation effort. On the other hand, the approach moreover clearly indicates the not-so-suitable for automation candidates, such as I and J.

Many test cases have dependencies on other test cases and common parts that could result automating several test cases at once with minor changes. These should be taken into consideration as shown in the real project example later in this paper. Last but not least, it is also possible to add new candidates to the coordinate system, and to remove existing candidates at any time, while keeping the prioritization list up-to-date. This aspect of the approach could prove to be a significant benefit for organizations using agile development processes, where specific requirements can be volatile and evolving, which will directly influence the test execution. It can also be applied to heavyweight projects such as safety-critical systems testing where new test cases are often added to the scope. In short, each test case i can be assigned an automation efficiency quotient

$$\eta_i = \frac{m_i}{a_i} \tag{1}$$

where m_i is the estimated manual effort for test execution and a_i is the estimated effort for automating the manual test. The bigger η_i, the better candidate test case i is for automation which explains the order shown on Fig. 1. The only perplexing point in this approach is how to estimate the manual and automation effort required for a specific test case.

IV. Generic Effort Estimation

Experienced professionals could eventually give their subjective estimations in a manner similar to the planning poker described in [2]. Since the automation effort is in its essence a development work, plenty of widely known estimation methods could be used, e.g. analogy-based estimation, parametric models, size-based estimation models, group estimations, mechanical, or judgmental combination. It is possible to apply some of these methods to manual testing effort as well, but the latter can also be assessed by a measure proposed in [1] called Execution Points (EP), which reflects the amount of work required to execute tests manually. Basically, this measure is based on the amount of

TABLE I
CONSIDERABLE FACTORS WHEN ASSESSING MANUAL TEST EXECUTION EFFORT

Factor	%
M1. Time needed for single manual test execution	20%
M2. Number of testing cycles and repeated executions per year	15%
M3. Repeating lots of actions to check simple difference on the final one	15%
M4. Multiple platforms, OS, browsers, etc.	15%
M5. Lots of different inputs	10%
M6. Large data inputs	10%
M7. Monotone repeatable actions that are easy to cause a human error or omission	5%
M8. Long preliminary setup or cleanup that could be avoided with automation	5%
M9. Lots of documenting and reporting that could be avoided with automation	5%

TABLE II
CONSIDERABLE FACTORS WHEN ASSESSING EFFORT FOR AUTOMATING MANUAL TEST CASES

Factor	%
A1. Time needed to implement automated testing	20%
A2. Complexity, including packaging, data and environmental challenges	15%
A3. Maintenance effort and code changes	15%
A4. Unstable requirements	10%
A5. Unstable application feature	10%
A6. Small code coverage increase	5%
A7. Test results bring little value to business	5%
A8. Unpredictable results when different output data is returned after each execution	5%
A9. Additional support from the development team	5%
A10. Test data generation or automated recovery needed	5%

test actions (user actions and observed results) found in a test specification, and takes into consideration the functional data, screen navigations, etc., and non-functional (use of network, etc.) characteristics of the applications and system environments exercised by the test actions. Whatever the selected method, assessing the accuracy of estimates is an important measure, so it is advisable to re-estimate test case candidates regularly.

In practice, only a relative effort estimation is required to prioritize test case candidates. Based on considerations in [1], [3], [5] and [8], as well as on the authors' experience, two tables containing important factors for effort estimation assessment were created, one for manual (Table I), and one for automated (Table II) testing.

Each table row contains an approximate percentage that gives basic idea about the impact of the relevant factor on the total effort for generic testing. However, this should serve only as an expectation-based and initial reference point, as it is based entirely on the authors' practical experience and will be very individual among different projects.

A. Assessing Manual Test Execution Effort

As Tables I and II show, time required is the most crucial factor for both manual execution and automation development (M1 and A1).

When estimating manual effort, one should devote a lot of attention on how often the test execution will be repeated (M2); on how many environments (M4); and how specific the test is (M3). The input data is also a very important factor in terms of size (M6), quantity and repetitiveness (M2, M3, M5, M7), especially when the test contains large data sets that utilize the same workflow but different data has to be input for each test run (e.g. data-driven and boundary tests with search, login or form submission). Time for setup, cleanup (M8), documentation and reporting (M9) could also be saved by simple automation script (e.g. creating or deleting lots of data entries or instances, maintaining action log, etc.)

B. Assessing Automated Test Execution Effort

With regard to test automation, complexity (A2) and maintenance costs (A3) play the second most important role in the estimation (after time needed for automation – A1). Their significance is complemented by the requirements stability (A4), i.e. whether the requirements are subject to significant changes. This can be calculated using requirements stability index (RSI). According to the Standish Group's 1995 Chaos Report, 73% of projects were either canceled or failed to meet expectations due to insufficient requirements definition and analysis. A 1997 study by Sequent Computer Systems reported that 76% of the 500 IT managers surveyed had been involved with failed projects at some point in their careers, and most failures were attributed to changing user requirements [9]. Another important factor is the application stability (A5) – frequent errors, crashes and failures are indicator of low stability. Put simply, the fewer changes are made to the software and the test environment, the less time will be spent on maintenance and code refactoring. This is further leveraged by code coverage increase from the relevant test case (A6). Automating specific software features helps businesses align their processes effectively with customer expectations and save time for manual work in other departments as the production of accurate information about the test results on time may have dramatic impact on business (A7). There are also fewer factors that affect business importance – mostly security, reliability, fast and easy obtaining of the test results and, if possible, generating the relevant test reports and documentation automatically.

In the end, one must consider some "tricky" parts of the automation: whether the test data is already generated or the automation will need to do that additionally (A10); whether there are different data outputs (e.g. search on constantly changing database, approximation algorithms, etc.), and is it easy to determine whether the output is plausible (A8); whether additional support from the development team will be needed (e.g. for setting unique names for controls and fields) (A9); whether recovery will be needed (A10) after the automation test (e.g. tests with big data freeze the server and restart is needed to continue the rest of the testing). All

these factors could reflect the positioning of a specific test case candidate in the coordinate system and that is why good assessment is very important. If applied properly, the approach suggested in this paper (referred as M/A Effort Approach from now on) can improve quality and response times, both the automation effectiveness and efficiency, and reduce the testing costs.

TABLE III
TEST CASES FOR ASSESSMENT

No.	Description	Dep.
T1	All buttons are visible when the app is started	-
T2	About button shows screen with current version and build.	T1
T3	Add Identity button is available and can be clicked.	T1
T4	Identity name is limited to 100 characters and displayed correctly.	T3
T5	Identity email is limited to 100 characters and displayed correctly.	T3
T6	Identity is inactive until a PIN is set.	T3
T7	Cannot enter PIN with less or more than 4 digits while adding identity.	T3
T8	After adding identity, user is able to login.	T6
T9	Cannot enter PIN with less or more than 4 digits while logging in.	T8
T10	Can login with the PIN entered from first attempt.	T8
T11	Can login with the PIN entered from second attempt.	T8
T12	Can login with the PIN entered from last(3rd) attempt.	T8
T13	Cannot login with the wrong PIN entered more than 3 times (identity blocked).	T8
T14	Can change PIN.	T6
T15	After adding 2 or more identities, a list is shown.	T3
T16	Screen starts scrolling after registering 5 identities.	T3
T17	Identity with invalid email address cannot be added (50 email addresses).	T3
T18	Identity with valid email address can be added (10 email addresses).	T3
T19	Identity with an already existing email address cannot be added.	T3
T20	Active identity is changed on click and shown on top.	T15
T21	Added identities can be deleted.	T3
T22	Identity remains in the list if the deletion is canceled.	T21

V. EMPIRICAL EVALUATION

It is not a common practice among the managers and team leaders to make prioritization based on manual and automation test effort. If any test prioritization is done, it is based mainly on priority of the test cases set by product owners. Many books about software quality assurance state exactly that automated tests should tackle the high-priority tasks first [10] [11] [12]. However, when properly applied, the M/A Effort Approach may overcome the "high-priority tests automated first" approach (HPTAF). To prove that, a comparison between those two approaches needs to be done.

As one of the authors works as software QA manager in a company developing authentication product, this product was used for empirical evaluation of the approaches. The product mainly consists of identity management and login using given identity with a PIN. Two different teams in the company developed identical mobile demo applications – one of the teams developed an Android application and one of the teams developed an iPhone application.

TABLE IV
TEST CASES ASSESSED WITH MANUAL EXECUTION FACTORS

No.	M1	M2	M3	M4	M5	M6	M7	M8	M9
T1	10	15	0	15	0	0	2	0	0
T2	5	5	0	5	0	0	0	1	0
T3	7	15	0	15	1	0	0	1	0
T4	15	5	12	5	5	5	2	3	0
T5	15	5	12	5	5	5	2	3	0
T6	12	10	3	5	2	0	1	2	0
T7	13	10	9	5	6	1	1	2	1
T8	2	15	3	15	2	0	1	3	0
T9	10	10	9	5	6	1	1	5	1
T10	2	15	3	15	2	0	0	5	0
T11	5	10	6	5	4	1	1	5	0
T12	9	10	9	5	6	2	2	5	0
T13	9	10	9	5	7	2	2	5	0
T14	15	10	3	5	4	1	1	4	0
T15	15	10	6	5	6	3	2	2	0
T16	18	5	12	5	9	5	3	2	1
T17	20	10	15	10	10	10	5	2	5
T18	19	15	12	10	8	7	4	2	1
T19	5	10	6	8	2	1	1	2	0
T20	10	10	3	5	1	0	1	4	0
T21	13	10	6	5	3	0	1	5	0
T22	3	5	6	5	1	0	0	4	0

TABLE V
TEST CASES ASSESSED WITH AUTOMATION FACTORS

No.	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
T1	15	5	10	8	5	1	1	0	5	0
T2	3	12	10	5	1	4	5	0	3	0
T3	3	1	5	1	2	3	1	0	0	0
T4	1	5	5	2	3	4	4	1	0	3
T5	1	5	5	3	3	4	4	1	0	3
T6	7	1	5	1	3	3	2	0	0	2
T7	3	1	10	10	5	5	4	1	0	2
T8	20	10	10	1	7	2	1	1	0	0
T9	3	1	10	10	5	5	4	1	0	1
T10	1	5	10	1	7	2	1	1	0	0
T11	1	5	10	1	7	3	2	1	0	0
T12	1	5	10	2	7	4	2	1	0	0
T13	5	10	15	3	7	2	2	1	0	5
T14	3	5	10	2	7	3	3	1	0	1
T15	10	5	5	5	5	2	3	2	0	5
T16	3	5	10	3	10	2	4	3	0	5
T17	1	10	10	3	3	4	4	3	0	5
T18	3	10	10	1	3	4	1	3	0	5
T19	2	10	5	1	3	4	2	1	0	2
T20	5	10	5	3	5	2	3	2	0	1
T21	3	5	5	3	1	3	4	3	0	4
T22	1	1	5	1	1	3	4	2	0	0

To test the mobile demo applications, a suite of 22 test cases (marked from T1 to T22) was created as shown in Table III. In the beginning, all of the tests were performed manually by two QA teams, each consisting of 7 experienced manual testers and 1 automation test developer. The software development process was Scrum and the automation framework used was Calaba.sh.

T1, T3, T8, T10, T18 were given high priority by the product owner. T2, T4, T5, T16 and T22 were given low priority. All the rest were given medium priority. High priority tests had to be executed 3 times a week, medium priority tests - 2 times a week and low priority tests - 1 time a week.

Each of the test cases was assessed with manual execution

factors (Table IV) and automation factors (Table V). Factor M1 assessment was based on average manual execution during previous test sessions. Factor M2 was calculated based on the priority given. Factors M3-M9 and all automation factors were based on team's expert knowledge and previous experience using technique similar to Scrum Planning Poker.

TABLE VI
TEST CASES WITH MANUAL EFFORT (m_i), AUTOMATION EFFORT (a_i) AND AUTOMATION EFFICIENCY QUOTIENTS (η_i) CALCULATED

No.	mi	ai	Quotient
T1	42	50	**0.840**
T2	16	43	**0.372**
T3	39	16	**2.438**
T4	52	28	**1.857**
T5	52	29	**1.793**
T6	35	24	**1.458**
T7	48	41	**1.171**
T8	41	52	**0.788**
T9	48	40	**1.200**
T10	42	28	**1.500**
T11	37	30	**1.233**
T12	48	32	**1.500**
T13	49	50	**0.980**
T14	43	35	**1.229**
T15	49	42	**1.167**
T16	60	45	**1.333**
T17	87	43	**2.023**
T18	78	40	**1.950**
T19	35	30	**1.167**
T20	34	36	**0.944**
T21	43	31	**1.387**
T22	24	18	**1.333**

Based on Tables IV and V, another table with the automation efficiency quotients was created (Table VI). Although the approach is based mainly on those quotients, dependencies between the test cases should also be considered. This is shown on Fig. 2.

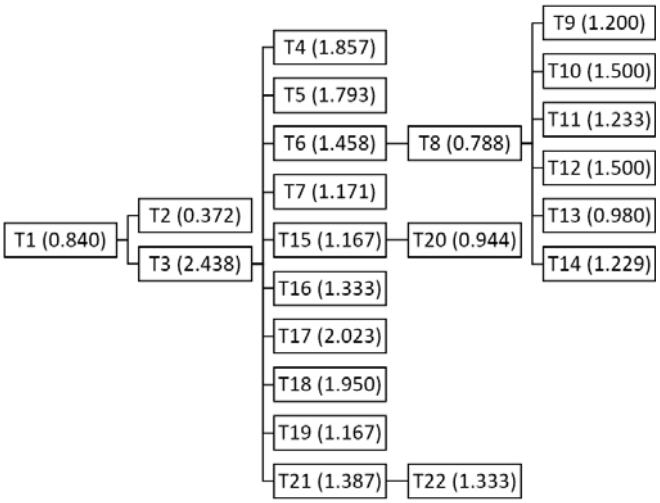

Fig. 2 Test cases with their dependencies and quotients

To compare both approaches, all the time spent on different projects, bug-fix retesting and meetings was excluded. Each working day was considered to have 6 productive working hours for manual testing which results 30 hours per week. All the time spent on automation by both team was averaged (e.g. if iOS team has spent 4 hours and the Android team has spent 6 hours in automation of the same test case, 5 hours were considered for each team in order to have better comparison of the approaches). The order of the tests' automation was preserved.

Table VII shows a week-by-week comparison of the both approaches in terms of hours spent in manual testing while showing the order of the test cases automated.

TABLE VII
COMPARISON

Week	Approach in this paper		HPTAF Approach	
	Man.(h)	Automated tests	Man.(h)	Automated tests
1	210	Preparation	210	Preparation
2	210	T1 (50%)	210	T1 (55%)
3	196	T1, T3 (65%)	196	T1, T3 (70%)
4	184	T3, T17 (45%)	184	T3, T6 (15%)
5	142	T17, T18	184	T6 (65%)
6	131	T4, T5	176	T6, T8 (15%)
7	123	T6, T21 (20%)	176	T8 (80%)
8	114	T21,T16(65%)	166	T8, T10 (60%)
9	108	T16,T22(65%)	155	T10, T18 (45%)
10	105	T22, T7 (75%)	113	T17,T18,T7(35%)
11	95	T7, T15 (40%)	93	T7, T9
12	84	T15	93	T11 (90%)
13	84	T19 (65%)	66	T11, T12, T13
14	77	T19, T20 (35%)	56	T14
15	70	T20, T8 (10%)	56	T15 (65%)
16	70	T8 (65%)	45	T15, T19 (45%)
17	60	T8, T10 (65%)	38	T19, T20(60%)
18	49	T10,T12(30%)	31	T20, T21(60%)
19	22	T11, T12, T13	22	T21, T2 (35%)
20	10	T14	20	T2, T4 (5%)
21	2	T9, T2 (35%)	9	T4, T5
22	2	T2 (90%)	9	T16 (65%)
23	0	T2	0	T16, T22
	2148		2308	total hours

Week 1 was spent in preparation of the test environment and full manual regression test session was performed (210 hours). Due to the dependencies shown on 2, Week 2 started with T1 automated for both teams. As the The M/A Effort Approach requires estimating, filling and updating Tables IV, V and VI, 4 hours for initial estimation and 1 additional hour per week were considered for that. Thus, HPTAF approach took small advantage during Week 2.

During Week 3 and Week 4, the advantage of HPTAF remained as both teams chose T3 (higher priority and higher $\eta_i = 2.023$ compared to T2). However, both The M/A Effort Approach and HPTAF finished with 184 hours manual effort per week after T1 and T3 were automated.

During Week 4 different tests were chosen for automation. T6 was chosen by HPTAF as next high priority test case in the role, while the M/A Effort Approach continued with T17 with $\eta_i = 2.023$. As T18 was very similar to T17 it was done during the same week (same was applied to HPTAF in Week 10).

As a result of the different automated test cases, the time spent for manual regression tests in Week 5 were different (142h for M/A Effort Approach vs. 184h for HPTAF). T4 and T5 were automated together (same was applied to HPTAF in Week 21 as these are low priority test cases).

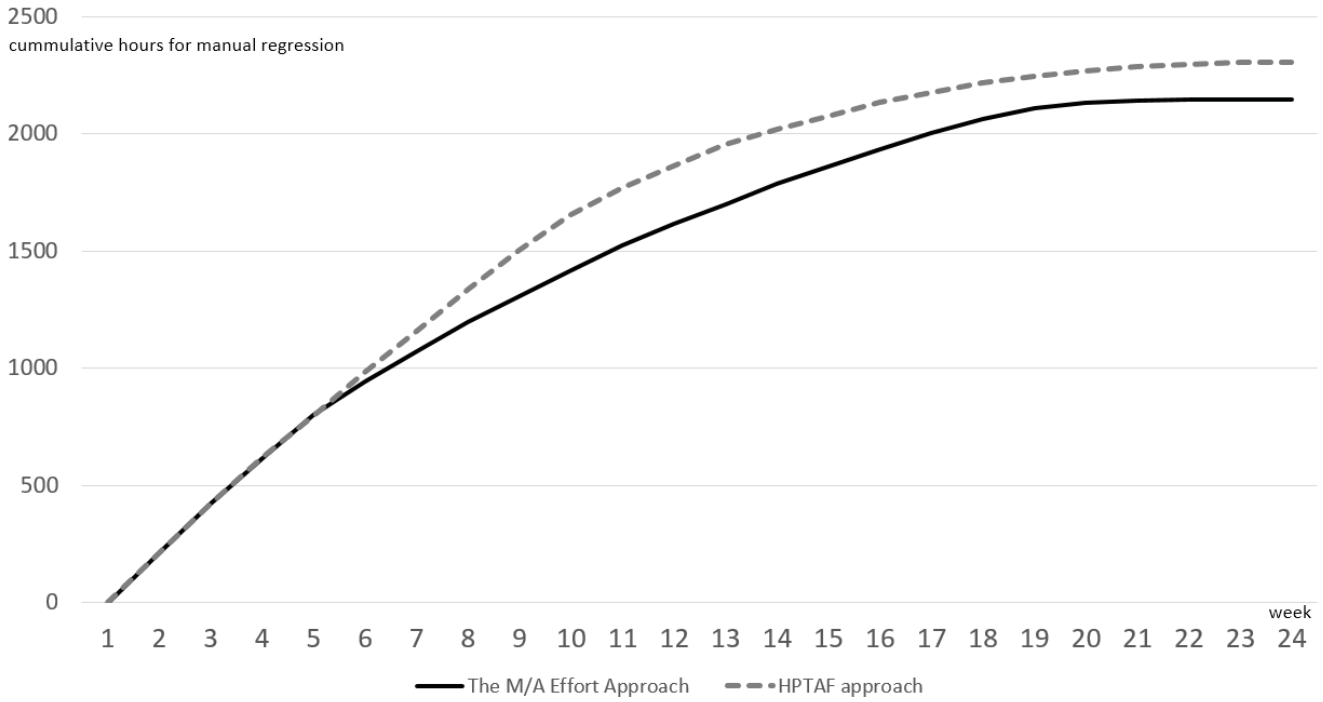

Fig. 3 Comparison between both approaches in cumulative man hours spent in manual regression tests

During Week 6 to Week 9, the difference between the both approach in time spent for manual regression tests become even bigger (108h for M/A Effort Approach vs. 155h for HPTAF) which means only 4 manual QA engineers were needed when using M/A Effort Approach while 6 QA engineers were needed when using HPTAF.

During Week 10, T17 and T18 were automated together, later during Week 13, T11, T12 and T13 were automated together again which resulted HPTAF to take just 56h compared to 70h for M/A Effort Approach.

The results were draw again in Week 19, after T11, T12 and T13 were automated together.

Up to Week 23, M/A Effort Approach took advantage over HPTAF again and the final result was 2308 cumulative hours for HPTAF approach compared to just 2148 cumulative hours for M/A Effort Approach as shown on Fig. 3.

The 160h saved represent more than five weeks of single person's manual work. This however is not the only benefit of the M/A Effort Approach. During the first weeks, more test cases were automated and more manual QA resource was freed.

Authors think that combining this approach with efficient methods for test case selection and minimization will result effective software testing that will contribute for better fault detection rate, and respectively – better software quality.

VI. Conclusion

This paper presents a simple approach to prioritize manual software test cases that are suitable for automation. It differs from existing approaches in that it gives an easy to follow method based on effort assessment. Factors that affect this assessment highly depend on the system itself and may be given different weights.

The suggested approach is very flexible because it allows working with a variety of assessment methods, and adding or removing new candidates at any time. The theoretical ideas presented in this article have been successfully applied in real world situations in several software companies by the authors and their colleagues including testing of real estate websites, cryptographic and authentication solutions, OSGi-based middleware framework that has been applied in various systems for smart homes, connected cars, production plants, sensors, home appliances, car head units and engine control units (ECU), vending machines, medical devices, industry equipment and other devices that either contain or are connected to an embedded service gateway.

Although the factors, percentages and quotient values provided in this paper's table can be changed, improved and modified as more teams use the approach in their test prioritization and more usage data is collected, the main idea of the approach remains unchanged.

Acknowledgments

This work is supported by the National Scientific Research Fund under the contract ДФНИ - И02/13.

References

[1] E. Aranha, "Estimating test execution effort based on test specifications", Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil, pp. 29-42, January 2009.
[2] M. Cohn, "Agile estimating and planning", Pearson Education, Prentice Hall, New York, USA, 2005.
[3] J. Fernandes and A. Di Fonzo, "When to automate your testing (and when not to)", July 2010.

[4] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey", King's College London, London, England, March 2012.
[5] B. Marick, "When should a test be automated", California, USA, 1998 [11th International Software Quality Week QualWeek 98 San Francisco, 1998].
[6] D. Mosley and B. Posey, "Just enough software test automation", Yourdon Press, Prentice Hall Professional, Englewood Cliffs, NJ, USA, 2002.
[7] V. Motwani, "The when and how of test automation", Bangalore, India (2001). [Annual International Software Testing Conference in India, 2001].
[8] G. Robert, "Automation selection criteria – picking the "right" candidates, LogiGear Magazine, 2014.
[9] M. Rouse, "What is requirements stability index (RSI)?", TechTarget, 2005.
[10] E. Dustin, J. Rashka and J. Paul, "Automated software testing". Reading, Addison-Wesley, 1999.
[11] S. Desikan and G. Ramesh, Software testing. Bangalore, India, Dorling Kindersley, India, 2006.
[12] S. Covey, A. Merrill and R. Merrill, First things first. Simon & Schuster, New York, USA, 1994.

Peter Sabev Peter Sabev is PhD student with strong interest in software test automation frameworks. Bronze medallist on Balkan Olympiad in Informatics as high school student, Peter has been a software developer, a documentation writer, a technical support specialist, and a project manager but his true passion lies in software testing and quality assurance. With over ten years of technologically focused experience in IT, Peter currently works as a QA manager for a cryptography company.

Katalina Grigorova Professor Katalina Grigorova is Head of Department of Informatics and Information Technologies at University of Ruse, Bulgaria. She received MSc degree in Applied Mathematics from Moscow Power Engineering Institute and PhD degree in Computer Aided Manufacturing from University of Ruse. Her research interests include Business and Software Architectures Modeling, Business Process Modeling, Automated Software Engineering, Databases, Data Structures and Algorithms Design, Programming. Prof. Grigorova is a member of Association of Information Systems (AIS) and its Bulgarian chapter BulAIS. She is a winner of IBM Faculty Award.