Int. j. inf. tecnol. (March 2024) 16(3):1967–1979
https://doi.org/10.1007/s41870-023-01476-4

ORIGINAL RESEARCH

The use of knapsack 0/1 in prioritizing software requirements and Markov chain to predict software success

Isaac Aduhene Armah[1] · James Ben Hayfron-Acquah[1] · Kate Takyi[1] · Rose-Mary Owusuaa Mensah Gyening[1] · Michael Eshun[1]

Received: 10 May 2023 / Accepted: 29 August 2023 / Published online: 15 September 2023

Abstract Requirements prioritization is one of the most valuable aspects of software engineering. This is primarily due to the fact that resources, be it time, skillset, or budget, are limited. Existing complex methodologies, such as analytical heuristic process (AHP) and planning game, face low adoption in the industry, promoting the need for more accessible techniques. This research introduces a novel contribution to software engineering by offering a simple and scalable approach to requirement prioritization (RP) and software acceptance prediction. The proposed approach consists of two key methods, knapsack 0/1 and Markov, to optimize RP and predict software acceptance respectively. By considering constraints, organizations can make enlightened decisions on handling requirements and optimize their minimum viable product. The results showcase significant time efficiency, with an average worst-case time of 5.645s for 10,000 requirements and an upper bound of 0.023s for the Markov prediction. This study aims to provide practitioners with a practical solution for prioritizing requirements and predicting software outcomes from user acceptance tests. By simplifying the process and offering compelling time complexity, this approach contributes to the enhancement of software development practices.

✉ Kate Takyi
takyikate@knust.edu.gh

Isaac Aduhene Armah
armahi@live.com

James Ben Hayfron-Acquah
jbha@yahoo.com

Rose-Mary Owusuaa Mensah Gyening
rmo.mensah@knust.edu.gh

[1] Department of Computer Science, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana

Keywords Knapsack 0/1 · Markov processes · Requirements prioritization · Software development · Testing

1 Introduction

Researchers have proposed many techniques aimed at efficient and effective software development. However, several fail due to resource limitations and incomplete requirements, amongst others [1]. From various works reviewed by [1], the proportion ranges from 20% to 85%. [2] also identified that the success rate of the software, based on the Standish Group report as of 2016, is 30%–40% and has inclined 16%–32% between 1994 and 2016. It is, therefore, imperative that further contributions are made to developing products compatible with user characteristics and specifications. The requirements stage in the Software development Life Cycle (SDLC) is, thus, essential to software development, as it tells organizations precisely what users want or need through requirements engineering (RE) and testing. SDLC is identified as a concept in software engineering that outlines the process involved in planning, developing, programming, testing, and implementing requirements of users [3]. The definition of [4] highlights the SDLC's main phases: planning, requirements analysis, design and prototyping, development, testing, deployment, and maintenance. Some online sources have fewer stated phases, but the definitions they provide are usually identified in the 7 phases. Generally, the expectation is that requirements should be met in chronological order. It made way for classic models such as the Waterfall model. Recent development processes have proven different and have made way for novel models. Tiranga and Sharma [4] makes an important observation that the latest development processes are contentious, resulting in very

uncertain requirements throughout the software development phases. Under such conditions, it is difficult to have a well-defined requirements documentation before commencing software development. This has brought about models like agile [5], which is refined to handle changing requirements. RE is identified as the process of obtaining valid requirements from stakeholders and end-users of a product [6]. This helps organizations to develop impactful software that helps end-users achieve their intended goals and objectives [7]. Testing is another key aspect of software development and may include Blackbox, Whitebox, and Greybox testing. Sawant et al. [8] defined software testing as a process that is carried out to assess a feature of a software to ascertain that it meets a set quality standard or user need. Software is built with a specification usually defined by users or business stakeholders; thus, testing is necessary to verify and validate that a software meets such specifications. Testing ensures that demands are rightly met and rejection by a client is reduced. One of the critical components of testing is acceptance testing (AT) also known as beta testing, being that it gives insight into users' reactions to a software or feature [9, 10]. Given the aforementioned problems with existing RE methods, this study proposes the use of knapsack 0/1 to classify requirements (using dynamic programming) and Markov to predict software success.

2 Literature

This section provides an overview and various related works relating to RP, testing, and the knapsack 0/1 problem.

References	Author(s)	Title	Objectives	Methodology	Results	Research gap
[11]	Sadiq et al. (2022)	An ensemble learning approach for the prioritization of software requirements	Present a method for the prioritization of SRs using rough set theory	Using the examination system of an educational institution	The proposed method captures the exact opinion of the decision-makers during the SRP process and no prior information is needed to compute the ranking values of the SRs	The proposed method has limited consideration of dynamic factors
[12]	Mohammad et al. (2021)	Exploring Stakeholder perspectives in fuzzy-attribute-based goal-oriented software requirement analysis	Present a fuzzy attributed goal-oriented software requirement analysis	FAGOSRA_MS approach	The proposed method considers the selection and prioritization of goals when multiple stakeholders participate in requirements analysis	
[13]	Leshob et al. (2020)	Optimizing software RP using the goal-oriented requirement language	Design a novel method to automate the RP process	Goal-oriented requirement language (GRL)	The proposed method presents the principles underlying the proposed RP method and discusses its possible implementation in practice	The proposed method requires stakeholders to understand the language before effectively participating
[14]	Sapunkov et al. (2019)	Automated user RP	Automate prioritization of user requirements	Fuzzy techniques: URPCalc approach for RP	Improved user RP for future and subsequent product deployments	
[15]	Tufail et al. (2019)	Comparative study of RP technique	Identify the optimal RP technique in a given context	Comparative analysis of RP techniques	Detailed analysis of the advantages and disadvantages of the top 10 RP techniques	Further research is needed on more RP techniques as the research was limited to the top 10 RP techniques

References	Author(s)	Title	Objectives	Methodology	Results	Research gap
[16]	Hudaib et al. (2018)	Comparison of RP techniques	Propose a general model for aiding in the selection of an appropriate RP technique for software requirements	Comprehensive analysis	The significance of prioritizing non-functional requirements. A general model for choosing an appropriate RP technique for software requirements	Additional review of other techniques is needed to improve the general model
[17]	Misaghian and Motameni (2018)	A tensor decomposition approach for RP	The effects of RP on functional, non-functional requirements and stakeholder views	Tensor decomposition	The proposed approach is 81% faster than AHP	Further research is needed on requirement dependencies. It was not possible to generalize the study to industrial scenarios due to the size of the dataset
[18]	Gupta and Gupta (2018)	Collaborative dependency-based requirement (CDBR) prioritization	Minimize disagreement and achieve a common opinion in the prioritization process	Particle swarm optimization, linguistic values, and execute-before-after (EBA) relation on requirements	The method addressed requirements interdependency, stakeholder communication, development team, and scalability issues in RP	
[19]	Hujainah et al. (2018)	A comprehensive review of software RP	Analysis and review of the RP domain	Review protocol and search strings based on research questions	Limitations in existing prioritization techniques such as complexity, dependency, scalability, etc	
[20]	Alzaqebah et al. (2018)	Enhancing the Whale Optimization Algorithm through hybridization with local search techniques	Order the requirement based on their importance and execution order	Combine automated reasoning techniques with user feedback	The proposed approach outperforms the AHP by approximately 40%	The proposed method converges to a suboptimal solution without exploring the entire solution space
[21]	Masadeh et al. (2018)	Dynamic adaptation and parameter control in the Grey Wolf Optimization Algorithm	Prioritize requirements that are given for a particular object	Meta-heuristic algorithm	The proposed approach performs better than the AHP in terms of time-consuming to rank the requirements	The proposed approach is sensitive to the initial values of its parameters
[22]	Hudaib et al. (2018)	Improving RP with the Whale Optimization Algorithm	Enhance RP by combining strengths of Whale Optimization of Algorithm (WOA) and Grey Wolf Optimization (GWO) Algorithm	Leverage the complementary strengths of the WOA and GWO algorithms for RP by providing a more robust and effective solution	The proposed method shows 91% accuracy and a 9% error rate in prioritizing the requirements	The proposed method requires domain-specific adaptation

References	Author(s)	Title	Objectives	Methodology	Results	Research gap
[23]	Sadiq et al. (2017)	Fuzzy set-based RP	Address inadequacies in prioritizing software requirements based on stakeholder needs	Fuzzy logic	Propose an efficient method for RP: identify stakeholders, classify requirements, and analyze stakeholders	The proposed method has not been tested on enough stakeholders
[24]	Morales-Ramirez et al. (2017)	Leveraging user feedback in tool-assisted prioritization of requirements with multiple criteria	Integrate automated reasoning techniques, adapted to incorporate information derived from user feedback	Multi-criteria-based RP method	Proposed elaborate decision-making solution using automated reasoning techniques whilst drawing into perspective the feedback users give	Further research is needed in scenarios in which one or more requirements are not associated with any feedback user
[25]	Keertipati et. al. (2016)	Feature improvement prioritization with app reviews	Identify features that improve software	Machine learning: regression ranking, weighted ranking, individual attribute	Identified key RP features: rating, user emotions, frequency	Further evaluations are needed in different app domains to understand the prioritization of features for improvements
[26]	Ur Rehman Khan et al. (2016)	RePizer framework for prioritizing software requirements	Order requirements based on specified conditions	Planning game, AHP	Works best with Planning Game in terms of improved accuracy and ease of use	
[27]	Liaqat et al. (2016)	Prioritizing requirement with majority voting technique	Address deadlock situations in RP	Focus on specific goals of stakeholders	proposed a technique (MVGB) that focused on deadlock situations in RP	Further research is needed on the dependency value of requirements
[28]	Mirjalili et al. (2016)	A nature-inspired optimization technique for solving complex problems	Solve optimization problems by finding the optimal solution or near-optimal solutions	Meta-heuristic algorithm	Improved quality of solutions and enhanced search process by leveraging exploration and exploitation phases	Further research is needed on additional techniques to ensure all constraints are satisfied
[29]	Achimugu et al. (2015)	Fuzzy multi-criteria decision-making (FMADM) for RP	Improve limitation problems on existing RP techniques	FMADM approach using local weight and collective global weight in a decision matrix	Improved efficiency of RP	Further research is needed to improve the efficiency of RP
[30]	McZara et. al. (2015)	Exploring linguistic tools and constraint solvers for software RP	Present a more time-efficient to implement for large-scale projects	SNIPR approach based on natural language processing	The proposed method consumes less time and improves selection accuracy	The approach does not consider all relevant constraints
[31]	Achimugu et al. (2014)	Review of research in software RP	Examine and assess current prioritization methods in the scope of defined research questions	Search for keywords such as requirements, RE, and prioritization	Limitations in techniques include coordination between stakeholders, requirement dependency, scalability, etc.	Data synthesis criteria cannot be guaranteed to be complete, robust, and free of publication bias

References	Author(s)	Title	Objectives	Methodology	Results	Research gap
[32]	Mirjalili et al. (2014)	A novel nature-inspired technique for optimization problems	Provide an effective and efficient algorithm to solve complex optimization problems	Meta-heuristic algorithm	Demonstrates competitive performance in terms of solution quality and robustness	Further research is needed to develop multi-objective versions of the GWO algorithm
[33]	Wiegers (2003)	Requirements of software	Prioritizes requirements by evaluating requirement values	Requirement cost and risks	The value of requirements cannot give distinct grounds to assess the properties of a requirement	

3 Methodologies

3.1 Implementation of the knapsack problem

The following algorithms are used to present the process involved in using the knapsack.

3.1.1 Algorithm prioritizing the requirements using knapsack

The goal is to maximize $\sum$ () $n\ i=1$ such that $\sum wrxr\ n\ r=1 \leq W$.

```
Start
  Define knapsack weight size W
  Define values of each item in array vr
  Define weights of each item in array WR
  GET length of values array n
  Define knapsack K
  DEFINE function knapsack (W,WR,vr,n):
    Initialize K[n,n] = 0
    For i in 0 and n + 1:
      For w in 0 and W + 1:
        If i = 0 or w = 0:
          Set K[i][w] = 0
        Else if WR [i-1] <= w:
          Set K[i][w] = max(vr[i-1] + K[i-1][w-WR[i-1]], and K[i-1][w])
        Else:
          Set K[i][w] = K[i-1][w]
        EndIf
      Endfor
    Endfor
    Return K[n][W];
  Endfunction
End
```

Algorithm for items in optimal solution set

```
Start
  Define K
  Define function optimal (n, W, WR, vr)
    While n > 0 and j > 0
      If K[i][j] != K[i-1][j]
        Print vr[i-1]
        SET J = W-WR[i-1]
        SET i = n-1
      Else
        SET i = n -1
      EndIf
    EndWhile
  EndFunction
End
```

3.1.2 Algorithm predicting acceptability using Markov chain

```
Start
  Initialize x,y   //x,y represent ratio in the initial state matrix
  DEFINE initial state matrix S[x,y]
  DEFINE transition matrix P[[a,b][c,d]]
  Define result set r[0,0]
  For i in range length(s)
    For j in range length p[0]
      result[i] += s[j] * p[j][i]
    EndFor
  EndFor
End
```

The knapsack's maximum weight is related to the strength or capacity of a team, time, and budget. The capacity of the knapsack can be calculated as the mean expressed as a percentage of these three constraints as shown in Eq. (1).

The following represents the formula for W:

$$W = \frac{time(t) + \text{budget}(b) + \text{capacity}(c)}{10} * 3^{-1} * 100 \quad (1)$$

where W is the total weight the knapsack can take, t is the expected time it will take to complete the project, b is the budget allocated to the project, and c *is* the size development team available to the project. A team is considered, making this approach adjustable to both large software firms that have several small teams and small firms which are usually made up of a few members.

The weight of an individual requirement is then found using the formula in Eq. (2)

$$wr = \frac{t + s + b}{10} * 3^{-1} * 100 \quad (2)$$

where WR is the value of an individual requirement, t is the time required to complete, b allotted budget for the requirement, and s is the average team skill. Since teams vary in size it is imperative to calculate this value as the average skill score of each member. As such, the team skill (s) as shown in Eq. 3 can be denoted as:

$$s = \sum n \cdot 10^{-1} + N^{-1} \quad (3)$$

where n is the 10-point score of each team member's skill and N is the total number of members in a team.

Knowing that the value of each requirement may vary based on various stakeholders and competitors, this study only suggests using value based on market study. This is to ensure that the value generated and agreed upon by management is realistic and not a guessed figure.

3.2 Prioritization using knapsack

For this study dynamic programming is modeled in a script to obtain prioritized requirements. This is because it is observed that using dynamic programming a pseudo-polynomial time can be achieved in solving the knapsack problem. Using naïve recursion, the knapsack problem has a time complexity of O(2n). On the other hand, using dynamic programming a time complexity of O(Kn) can be achieved. Making use of dynamic programming makes it easy as well. The knapsack algorithm is made of two parts, populating the knapsack and finding the items in the optimal solution.

3.3 Predicting the acceptance using Markov chain

The next step is to make use of the Markov chain to predict the acceptability of the software product from users. It is used in this study because it is a simple stochastic method, that can adequately be used to predict the state or value of a variable. Its use in this study aims to provide stakeholders insight as to whether their software product would thrive in the market. The initial state matrix, which is used as the current market share is defined using $So = [A, A\prime]$.

Where A is the current ratio for an organization A and $A\prime$ is the ratio against the same given organization. It can be likened to the competitor's current ratio in the market.

A transition matrix (T) in Eq. (4) is also defined using the test results of an organization and its competitors. This is expressed as:

$$T = \begin{bmatrix} aa\prime \\ a\prime a \end{bmatrix} \quad (4)$$

where 'a' is the resulting ratio in favor of a which is the organization and $a\prime$ is the resulting ratio in favor of competitors.

After evaluating using the dot product of the initial state matrix (S_0) and a transition matrix (P), next state (S_n) matrices can be obtained. These are denoted by $S_n = S_0 * P$.

The final state or stationary matrix (S) is defined using $S = S * P$ (Eqs. 5 and 6). The stationary matrix denotes the likely eventual outcome at the current market ratio. It is important to note that the test used in this study is a UAT test, as it provides insights into users' reactions to software in reality.

Given that $S = [A_1, A_2]$ and $P = \begin{bmatrix} p_1p_3 \\ p_2p_4 \end{bmatrix}$, the stationary matrix evaluates as

$$[A_1A_2] * \begin{bmatrix} p_1p_3 \\ p_2p_4 \end{bmatrix} = [A_1A_2]$$

$$[(A_1p_1 + A_2p_2), (A_1p_3 + A_2p_4)] = [A_1A_2]$$

$$A_1p_1 + A_2p_2 = A_1 \quad (5)$$

$$A_1p_3 + A_2p_4 = A_2 \quad (6)$$

Knowing that the sum of a probability ratio (Eq. 7) is 100% or 1 then, $A_1 + A_2 = 1 OR 100\%$.

$$A_1 + A_2 = 1 \quad (7)$$

therefore,
$A_1 = 1 - A_2$ and $A_2 = 1 - A_1$

To simplify the equation, either Eqs. (5) and (7) or Eqs. (6) and (7) are solved using substitution. For instance, using Eqs. (5) and (7), by substitution Eqs. (5) becomes (8) represted as:

$$(1 - A_2)p_1 + A_2p_2 = 1 - A_2 \tag{8}$$

3.4 The proposed framework

The proposed technique suggested in this study is illustrated in Fig. 1. To strengthen the development process requirements are classified by treating them as a knapsack 0/1 problem. The knapsack problem can be described in lay terms as a problem where a person has a knapsack of fixed size (W) and items (r) of varying values (VR) and varying corresponding weights/sizes (WR), for which the person must find the best combination (in terms of value) of items that will fit the *"knapsack"*. Various forms of this problem exist. However, concerning this study, 0/1 knapsack is chosen. The reason is that, with a 0/1 knapsack, a person is restricted to selecting just a single item. This scenario is especially true for software requirements. Approaching the requirement halfway is not an ideal situation given that in the context of software most projects are already failing. Given the context of a software project, with its requirements and resource constraints, this study utilizes a simple dynamic program to ascertain the items that value the optimal value and weight given the knapsack weight. The set of items returned can be regarded as the most pressing or valuable requirements to tackle for a software project to be successful.

The next step proposes the use of the Markov chain to predict the software's acceptance. By using user acceptance test (UAT) results (as a ratio) and considering current market competition (as a ratio) as data for the Markov this can be achieved. A Markov chain makes use of an initial state which is set to the market share of the organization or team undertaking the software project in context and its competition share. The transition matrix is defined using the UAT results as metrics. The obtained values are then used in Markov chain multiplication which then produces various resulting sets over various iterations until a final state (stationary matrix) is achieved. Given that the program was run on a computer, a control matrix was defined to ensure the program's function does not run infinitely. This is set to a value of the minimum difference in growth to what a company considers significant to its operations. the final state matrix given that the UAT does not change over a period can be used as the change in market share the organization can expect. The transition matrix can be changed when the UAT changes to make up for any changes that may occur to have more relevant results given a specific period. Figure 1 shows the added steps proposed in this study to the software development process. Every main step on the left with a corresponding branch to the right, the main step is visited before the right branch.

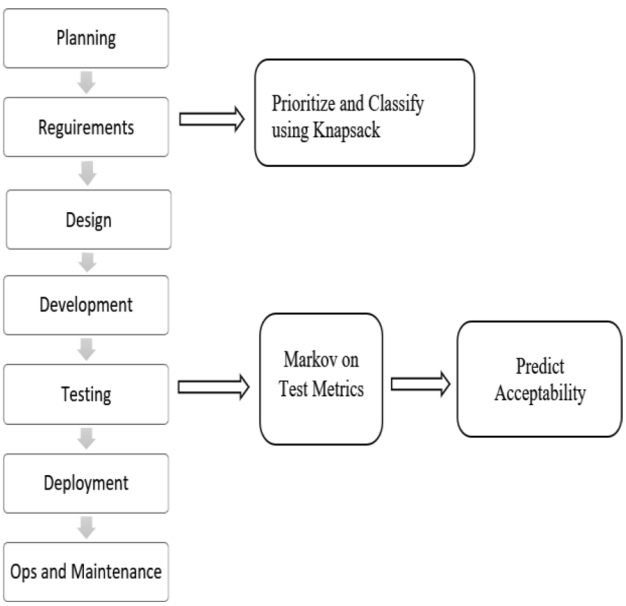

Fig. 1 Illustration of the proposed implementation

4 Implementations

As stated earlier, the Knapsack was used to classify and prioritize requirements whilst the Markov was used as a predictive model to determine the acceptability of software. The methods were implemented in code using Python (3.9) script using Pycharm Community IDE. Using a sample of 100 requirements, both individual weights (WR) and their corresponding values(VR) were generated in Eq. (9) using the code below:

$$\text{int}((random() * 100 + 1).\_\_floordiv\_\_(1))\, for\ i\ in\ range(100)] \tag{9}$$

This formed a random data sample to be used to test the functions in this study. Random data was used to mimic real-life data because it is heterogeneous. Using the knapsack defined in the previous section, the items in the optimal solution can be obtained from the populated knapsack. These items serve as the requirements that offer the best possible combination of requirements that give the highest value, given the various constraints.

4.1 Obtaining the maximum possible value

The goal of the knapsack algorithm is to provide the best combination of items that yield the highest value given a constraint, in this case, a knapsack weight. The individual weights and values used in this study are expressed as a

percentage. As such the maximum knapsack size is given as a percentage as well.

4.2 Predicting acceptability using the Markov

Markov is implemented using a hypothetical scenario where company A has a market share of 30%. Given this premise, the competition is regarded as occupying the remaining 70% of the market share. This forms the initial state matrix for the Markov ($S = [0.3, 0.7]$). The transition matrix (P) used is denoted in Eq. (10) as follows:

$$P = \begin{bmatrix} 0.80.2 \\ 0.60.4 \end{bmatrix} \tag{10}$$

From the matrix above, when the UAT score of A is 0.8, 0.2 is allotted as the chance of rejection or migration from the product to the competitor. Whilst on the second row, a 0.6 chance indicates, that the chances of acceptance at the moment stand at 0.6 for competitors, whilst 0.4 represents the chances of them (competitors) losing their current users. To ensure the script does not run infinitely, a control variable was introduced. The variable is assigned a value that acts as the desirable difference at which the initial state matrix differs from the next state or final state.

4.3 Parameters used in knapsack

Classifying RP as a knapsack problem means the variables/conditions necessary for a valid knapsack situation must exist. The variables (time, budget, team strength/capacity) were considered in determining the maximum weight of the knapsack. Each variable is valued using a 10-point-scale system (minimum = 1, maximum = 10), and a mean of these variables expressed as a percentage is evaluated to arrive at the final knapsack weight. The following was deduced as the formula for arriving at the maximum weight, W in Eq. (11).

$$W = \frac{time(t) + \text{budget}(b) + \text{capacity}(c)}{10} * 3^{-1} * 100 \tag{11}$$

where W is the total weight the knapsack can take, t is the expected time it will take to complete the project, b is the budget allocated to the project, whilst c *is* the size/capacity of the team available to the project.The variable c is considered to make room for this approach's usage in small firms. To complete the knapsack problem, the individual weight of requirements (items) and their corresponding values need to be identified as well. Hence, factoring in simplicity, the weight can be scored using a 10-point-scale system expressed as a percentage as shown in Eq. (12).

$$wr = \frac{t + s + b}{10} * 3^{-1} * 100 \tag{12}$$

Another important aspect considered in this study is that team size may vary based on the organization. As such, to adequately find the team capacity, it is important that, the average skill set of the team (Eq. 13) is considered. This can be expressed as

$$s = \sum n \cdot 10^{-1} + N^{-1} \tag{13}$$

where n is the 10-point score of each team member's skill and N is the total number of members in a team.

Because the value of a requirement also varies per organization and project, this study does not attempt to provide a means to value individual requirements. Various research provides methods that suggest means to achieve that. What this suggests is to value requirements based on market analysis since companies in the software development sphere barely operate without competition. This will ensure that the value generated is unbiased to stakeholder specialty, preference, or amongst other limitations.

4.4 Predicting the next state and final outcome

To have some control over what marginal difference is ideal for each project or institution, a control variable is introduced and assigned a value to differentiate between a point in the initial matrix and subsequent state matrices generated. This also ensures the Markov function has a definite ending point. Python script is run iteratively to predict the outcome which in this case is the stationary matrix. This is the final possible outcome given the conditions above. Using Eqs. (11) and (13) from the previous section, we have:

$$(0.3 * 0.8) + (0.7 * 0.6) = A_1$$

Therefore, $A_1 = 0.66$

For the next state matrix $S_1 = [A_1, A_2], A_1 + A_2 = 1$. Hence $A_1 = 1 - A_2, i.e 1 - 0.66 = 0.34$. Thus state 1 or the first state $S_1 = [0.66, 0.34]$.

For the final state matrix, Eq. (8) is used,

- $(1 - A_2)p_1 + A_2 p_2 = 1 - A_2$
- $(1 - A_2) * 0.8 + (A_2 * 0.6) = 1 - A_2$
- $0.8 - 0.8A_2 + 0.6A_2 = 1 - A_2$
- $0.8 - 0.2A_2 = 1 - A_2$
- $A_2 - 0.2A_2 = 1 - 0.8$
- $0.8A_2 = 0.2$
- $A_2 = 0.25$

If $S = [A_1, A_2]$ and Eq. (13) holds then $A_1 = 1 - 0.25 = 0.75$. Then the final state matrix from the given scenario is $S = [0.75, 0.25]$.

5 Results and discussions

In this study, randomly generated data is utilized for each dataset in the knapsack, and scenarios are tested using assumptions to examine the Markov chain process. Table 1 presents the randomly generated data using a 100×100 dataset. The best combination of items that fit the knapsack had a total value of 9768. It is observed from the generated data that the various requirements (r) have significantly large values (VR) individually and, yet not all these requirements have reasonable weights (WR). If these requirements are analyzed individually, it is easy to see that making the right choices in prioritizing is not a straightforward task and certainly considering the individual value alone isn't sufficient. For instance, from Table 1, requirement 94 (r94) has a value of 1000, yet it weighs 80 leaving little room for other requirements to be considered. Supposing r22 is taken, with the second highest value of

Table 1 Randomly generated weights and values of requirements

r	*WR*	*VR*	*r*	*WR*	*VR*	*r*	*WR*	*VR*
R1	37	605	R41	54	257	R81	11	382
R2	31	472	R42	9	257	R82	5	193
R3	42	566	R43	39	583	R83	4	270
R4	61	93	R44	26	800	R84	74	579
R5	44	512	R45	19	155	R85	84	106
R6	38	333	R46	74	416	R86	80	457
R7	56	943	R47	11	691	R87	4	663
R8	96	311	R48	66	396	R88	45	211
R9	12	945	R49	44	951	R89	20	693
R10	1	320	R50	5	153	R90	43	268
R11	6	408	R51	11	952	R91	81	564
R12	49	286	R52	2	263	R92	49	22
R13	88	328	R53	37	816	R93	59	290
R14	15	480	R54	87	103	R94	80	1000
R15	69	656	R55	35	930	R95	2	93
R16	88	509	R56	98	636	R96	65	9
R17	53	42	R57	41	654	R97	69	117
R18	98	962	R58	19	141	R98	8	962
R19	2	926	R59	12	325	R99	12	67
R20	26	246	R60	95	720	R100	29	254
R21	80	527	R61	41	873			
R22	17	976	R62	67	321			
R23	35	956	R63	30	828			
R24	64	426	R64	42	705			
R25	74	316	R65	43	569			
R26	53	497	R66	5	319			
R27	47	384	R67	7	985			
R28	57	897	R68	7	276			
R29	44	990	R69	24	530			
R30	14	272	R70	43	981			
R31	6	995	R71	83	115			
R32	69	868	R72	62	552			
R33	64	570	R73	92	889			
R34	56	524	R74	86	844			
R35	27	527	R75	48	289			
R36	88	834	R76	77	526			
R37	52	285	R77	28	375			
R38	89	747	R78	92	138			
R39	95	727	R79	81	567			
R40	66	831	R80	16	501			

Table 2 Optimal requirements weights and values pair

r	WR	VR
10	1	320
95	2	93
52	2	263
19	2	926
87	4	663
83	4	270
66	5	319
31	6	995
11	6	408
67	7	985
98	8	962
51	11	952
47	11	691
9	12	945
22	17	976

Table 3 Average runtime of knapsack and optimal items selection

Dataset	Knapsack			
	Knapsack function		Optimal items	
	perf_counter	timeit	perf_counter	timeit
100	0.00482177	0.009266862	7.002E−05	6.9792E−05
1000	8.881735E−02	0.4757422	3.98E−04	0.0048617
10,000	0.6658636	5.6483326	3.80E−03	0.0224439

Table 4 Average runtime of Markov function

Markov	
Markov function	
perf_counter	timeit
1.66917E−05	0.022904308

Iteration 1 result is [0.6599999999999999, 0.33999999999999997]

Iteration 2 result is [0.7319999999999999, 0.26799999999999996]

Iteration 3 result is [0.7463999999999998, 0.25359999999999994]

Iteration 4 result is [0.7492799999999998, 0.25071999999999994]

Iteration 5 result is [0.7498559999999999, 0.25014399999999999].

Fig. 2 The result of the Markov matrix at various iterations

976, the result total value is 1976 (i.e., the sum of values of r22 and r94), and a weight of 97 (i.e., sum weights of r22 and r94) is accumulated since r22 weighs 17. This leaves the knapsack at 97%.Very few requirements will fit the remaining space and from the generated data, such data won't be near 9768.

Table 2 shows the best weight-value pairs that make up the optimal value for the knapsack. Compared to the previous comparison discussed, the total knapsack weight is 98 from Table 2, yet the total value is 9 even at such great value, there is still room in the knapsack.

In predicting acceptability, there is a less than 0.01 difference between the manual calculations and the computation done through a computer. In the previous section, $S = [0.75, 0.25]$, whilst Fig. 2 shows S rather approaching [0.75,0.25].

The resulting matrix from Markov provides companies with information as to too much growth to expect given that the conditions for the variables remain the same. In that case, a company has the opportunity to either pivot or go ahead with further development of a product. A company also has the benefit to revisit requirement values, to ensure specific requirements that users find useful are adequately valued since it is such requirements that can cause a user to move to or from a competitor.

To reduce biases with execution time measurement, two unique time functions in Python were utilized in measuring the various functions that were run. Furthermore, the functions were run several times during measuring the execution time, as every single run varied in time; and then an average was struck off the various runs, to attain a more relative time. The functions were executed 10 times both manually and automatically. The results of these tests are summarized in Tables 3 and 4

In order to compare with existing work, a sample dataset of size 20 was generated for testing. This was done because the ReproTizer tool as suggested by made use of an input size of 20, hence ensuring fairness. The results of this comparison are indicated in Table 5.

Table 5 Comparing the runtime with ReproTizer and optimal items

Algorithm	Input dataset size	Average runtime (s)
ReproTizer	20	0.4
Knapsack	20	0.0118295
Optimal items	20	9.1999999999980e-05

6 Implications of the study

This study has aimed to provide a simple yet scalable means of prioritizing requirements using simple tools and techniques such as dynamic programming and Markov chain. This has been done by first classifying RP as a knapsack 0/1

problem. By so doing, dynamic programming can be quickly utilized in code, to find fill a "knapsack" given the necessary constraints and provide items in the optimal solution of the knapsack programmatically. the reason for this is identified in the study, that there are complexities with some existing methods such as AHP, which prevents more adoption of such techniques, given also that it was not scalable. From the research findings, the software still has a low success rate for many reasons, and as such it validates the importance of simplified methods, which can be quickly adopted by the industry to enhance the development process. The use of Markov in this study has been as a predictive tool that would enable firms adequately predict the success of their application based on the results of their beta test and factors surrounding their competitors. With this, measures can be put in place pre-deployment, or projects can be pivoted to reduce risks of low acceptance from users. RP in itself has its benefits, but utilizing a predictive model with it would further enhance it by providing insights that inform stakeholders if they are on the right track.

7 Limitations of the study

This research does not implement a special algorithm (software) that evaluates requirements based on market demand, competition, and stakeholder needs or business processes automatically. Requirements are generated according to stakeholders' demands, what resources they have, and what they perceive as most important. It would be much simpler or ideal to add value to requirements using an Artificial Intelligence (AI) module which will study the market and give suggested values to requirements. Also, the knapsack 0/1 and Markov analysis used in this research are not used in machine learning, artificial intelligence, or software product to make the necessary deductions. It is only implemented in simple programs which accept the inputs and produces the results. No charts or reporting tool is implemented to provide management-level insight. The research does not include test cases.

8 Conclusions and recommendations

The value of requirements cannot be taken lightly, especially in an environment where users want their demands met quickly and where there is stiff competition. Meeting and prioritizing requirements rightly could spell the difference between a successful company and an unsuccessful one. This being known, many researchers have proposed many ways to enhance development by prioritizing/ranking requirements. These include genetic algorithms and novel meta-heuristic algorithms of which some have been reviewed in this study. This study however proposes the use of the knapsack 0/1 problem. Weights for the knapsack are based on factors that hinder the success of software such as time, skill, and resource available expressed as a ratio. The corresponding values are proposed to be based on not just a business perspective which can be biased, but this study proposed the use of actual market data. Given that many softwares have been subject to failure, this study further suggests the use of the Markov chain as a predictive utility to help the necessary stakeholders of a project. Having implemented the proposed solution of this study using various datasets (100, 1000, 10,000), the runtimes showed great promise. The runtime ranged from 0.66558636 s (s) to a maximum of 5. 6483326 s average for 10 consecutive execution cycles for a dataset of 10,000. Also, the runtime of the Markov function has an average of 0.022904 s after 10 consecutive execution cycles. Having obtained reasonable runtimes with comparatively simpler tools, it can be concluded that organizations have simpler tools to use during development. At the ending the study, it is recommended for the advancement of the work to have an algorithm that does market analysis to ensure the right values are given to requirements. This ensures less bias towards value assignments and makes it easier for teams to gather such information from the market. The proposed work could also be enhanced to read data inputs from a file for knapsack values. Using random data made generating a huge set of requirements seamless. However, in the real world, these values can only be entered manually given the current state of the study. It is therefore highly recommended as an efficient method, that the script can read values from an Excel file or similar after the requirements are gathered with their weights and values assigned to them. It is also noticed that setting weights as percentages gave some limit to the number of requirements. It is proposed that in further study, the individual weights are further reduced to get more items into the knapsack or new ways can be employed to get more requirements to fit the optional solution set of the knapsack by relatively reducing the weights.

Acknowledgements We want to thank the Department of computer science, KNUST for their support and encouragement which contributed to the success of the research conducted.

Author contributions All authors were involved equally in the preparation and writing of the manuscript, review of the manuscript, experimentations, implementations, and analysis of results. The order and appearance of author names do not bear any effect on their contribution.

Funding This research did not receive any funding or grant from any organization, person, or body.

Availability of data and materials This does not apply to the research conducted.

Declarations

Conflict of interest We declare that there are no competing interests concerning the research conducted. Non-financial interests: The research conducted bears no financial interests from any organization, persons, or bodies.

Ethical approval This does not apply to the research conducted.

References

1. Cerpa N, Bardeen M, Kitchenham B, Verner J (2010) Evaluating logistic regression models to estimate software project outcomes. Inf Softw Technol 52(9):934–944. https://doi.org/10.1016/j.infsof.2010.03.011
2. Mehr Awan K, Ikram Lali M, Shehzad B, Mehr Awan K, Ikramullah Lali M, Aslam W (2017) Identification of patterns in failure of software projects. J Inf Sci Eng 33:1465–1479. https://doi.org/10.6688/JISE.2017.33.6.5
3. Sharma MK (2017) A study of SDLC to develop well-engineered software. Int J Adv Res Comput Sci 8(3):520–523
4. Tiranga SK, Sharma N (2019) Nimble agile a new approach in software testing. Int J Res Anal Rev 6(2):904–908
5. Nundlall C, Nagowah SD (2021) Task allocation and coordination in distributed agile software development: a systematic review. Int J Inf Technol 13:321–330. https://doi.org/10.1007/s41870-020-00543-4
6. Coutinho JCS, Andrade WL, Machado PDL (2019) Requirements engineering and software testing in agile methodologies: a systematic mapping. ACM Int Conf Proc Ser. https://doi.org/10.1145/3350768.3352584
7. Sawyer P, Sommerville I, Viller S (1999) Capturing the benefits of requirements engineering. IEEE Softw 16(2):78–85. https://doi.org/10.1109/52.754057
8. Sawant AA, Bari PH, Chawan P (2012) Software testing techniques and strategies. J Eng Res Appl 2(3):980–986
9. Mishra DB, Panda N, Mishra R et al (2019) Total fault exposing potential based test case prioritization using genetic algorithm. Int J Inf Technol 11:633–637. https://doi.org/10.1007/s41870-018-0117-0
10. Jamil MA, Arif M, Abubakar NSA, Ahmad A (2017) Software testing techniques: a literature review. In: Proceedings—6th international conference on information and communication technology for the muslim world, ICT4M 2016, 177–182. https://doi.org/10.1109/ICT4M.2016.40
11. Sadiq M, Devi VS (2022) A rough-set based approach for the prioritization of software requirements. Int J Inf Technol 14:447–457
12. Mohammad CW, Shahid M, Hussain SZ (2021) Fuzzy attributed goal-oriented software requirements analysis with multiple stakeholders. Int J Inf Technol 13:1–9
13. Leshob A, Hadaya P, Renard L (2020) Software requirements prioritization with the goal-oriented requirement language. In: Chao KM, Jiang L, Hussain O, Ma SP, Fei X (eds) Advances in E-business engineering for ubiquitous computing. ICEBE 2019. Lecture notes on data engineering and communications technologies, vol 41. Springer, Cham. https://doi.org/10.1007/978-3-030-34986-8_13
14. Sapunkov A, Afanasieva T (2019) Software for automation of user requirements prioritization. ACM Int Conf Proc Ser Part F1482:1–5. https://doi.org/10.1145/3318236.3318251
15. Tufail H, Qasim I, Masood MF, Tanvir S, Butt WH (2019) Towards the selection of optimum requirements prioritization technique: a comparative analysis. In: 5th international conference on information management, ICIM 2019, pp 227–231. https://doi.org/10.1109/INFOMAN.2019.8714709
16. Hudaib A, Masadeh R, Qasem MH, Alzaqebah A (2018) Requirements prioritization techniques comparison. Mod Appl Sci 12(2):62. https://doi.org/10.5539/mas.v12n2p62
17. Misaghian N, Motameni H (2018) An approach for requirements prioritization based on tensor decomposition. Require Eng 23(2):169–188. https://doi.org/10.1007/s00766-016-0262-6
18. Gupta A, Gupta C (2018) CDBR: a semi-automated collaborative execute-before-after dependency-based requirement prioritization approach. J King Saud Univ Comput Inf Sci. https://doi.org/10.1016/j.jksuci.2018.10.004
19. Hujainah F, Bakar RBA, Abdulgabber MA, Zamli KZ (2018) Software requirements prioritisation: a systematic literature review on significance, stakeholders, techniques and challenges. IEEE Access 6:71497–71523. https://doi.org/10.1109/ACCESS.2018.2881755
20. Alzaqebah A, Masadeh R, Hudaib A (2018) Whale Optimization Algorithm for requirements prioritization. In: 2018 9th international conference on information and communication systems, ICICS 2018, 2018-Janua, pp 84–89. https://doi.org/10.1109/IACS.2018.8355446
21. Masadeh R, Alzaqebah A, Hudaib A (2018) Grey Wolf Algorithm for requirements prioritization. Mod Appl Sci 12(2):54. https://doi.org/10.5539/mas.v12n2p54
22. Hudaib A, Masadeh R, Alzaqebah A (2018) WGW: a hybrid approach based on Whale and Grey Wolf Optimization Algorithms for requirements prioritization. Adv Syst Sci Appl 02
23. Sadiq M (2017) A fuzzy set-based approach for the prioritization of stakeholders on the basis of the importance of software requirements. IETE J Res 63:616–629. https://doi.org/10.1080/03772063.2017.1313140
24. Morales-Ramirez I, Munante D, Kifetew F, Perini A, Susi A, Siena A (2017) Exploiting user feedback in tool-supported multicriteria requirements prioritization. In: Proceedings—2017 IEEE 25th international requirements engineering conference, RE 2017, pp 424–429. https://doi.org/10.1109/RE.2017.41
25. Keertipati S, Savarimuthu BTR, Licorish SA (2016) Approaches for prioritizing feature improvements extracted from app reviews. ACM Int Conf Proc Ser. https://doi.org/10.1145/2915970.2916003
26. Ur Rehman Khan S, Peck Lee S, Dabbagh M, Tahir M, Khan M, Arif M (2016) RePizer: a framework for prioritization of software requirements. Front Inform Technol Electron Eng 17(8):750–765. https://doi.org/10.1631/FITEE.1500162
27. Liaqat RM, Ahmed MA, Azam F, Mehboob B (2016) A majority voting goal based technique for requirement prioritization. In: 2016 22nd international conference on automation and computing, ICAC 2016: tackling the new challenges in automation and computing, pp 435–439. https://doi.org/10.1109/IConAC.2016.7604958
28. Mirjalili S, Lewis A (2016) The Whale Optimization Algorithm. Adv Eng Softw 95:51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008
29. Achimugu P, Selamat A, Ibrahim R (2015) Using the fuzzy multicriteria decision-making approach for software requirements prioritization. Jurnal Teknologi 1:1–6
30. McZara J, Sarkani S, Holzer T et al (2015) Software requirements prioritization and selection using linguistic tools and constraint solvers a controlled experiment. Empir Software Eng 20:1721–1761. https://doi.org/10.1007/s10664-014-9334-8
31. Achimugu P, Selamat A, Ibrahim R, Mahrin MNR (2014) A systematic literature review of software requirements prioritization research. Inf Softw Technol 56:568–585. https://doi.org/10.1016/j.infsof.2014.02.001

32. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey Wolf Optimizer. Adv Eng Softw 69:46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007
33. Wiegers KE (2003) Software requirements, 2nd edn. Microsoft Press, p 544