ISSN 2278-3091
Volume 3, No.1, January – February 2014
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://warse.org/pdfs/2014/ijatcse01312014.pdf

A Decision Engineering Framework for Scrum Autonomy

Aditya Falodiya[1], Ajay Jain[2]
[1]Sr. Engineering Manager, Adobe Systems, USA,aditya@adobe.com
[2]Quality Engineering Manager, Adobe Systems, India, ajjain@adobe.com

ABSTRACT
In an agile based software product development process, teams should act in autonomous way to be more effective. This can be achieved by providing more empowerment to scrum teams in day to day decision making. It is also observed that scrum teams generally focus highly on feature delivery. Although bugs or defects in system are addressed during the feature development process but due to feature development getting priority, bugs are often ignored and accumulated to be taken up only after feature development. This paper proposes a process model and design of a system to assist agile teams to be more autonomous with their decision making on resources and efforts while ensuring a high quality sprint.

Keywords
Agile, Scrum, Decision Engineering.

1. INTRODUCTION

One of the critical aspects for a high performance agile team is empowerment of scrum [1] teams so that they can function as autonomous teams. Such teams are known as self-organizing teams [2]. It is also observed that teams are more effective if they are self-managing and takes decision themselves rather than relying on management team especially for day to day operations.

It is observed that in an agile software product development, scrum teams generally focus highly on feature delivery. This is also true for most of the product development teams working on their first release as they need to churn out features at a rapid pace.

Although bugs or defects in system are addressed during the feature development process but due to feature development getting priority, bugs are often accumulated in a sprint. It is only during last leg of sprint, they are handled with more focus and subsequently due to various constraints either features are delivered partially or deferred due to bugs in system.

Various project and scrum metrics like feature velocity, earned value, burn down etc. are also related to the feature or tasks in hand. There are some established metrics that helps in gauging the extent of defects in the system but they do not help scrum master effectively in taking decisions to guide team to focus on feature development or bug fixing at any point during the development process.

Generally, all day to day scrum operations and decisions are taken by engineering management based on heuristics, domain knowledge and criticality of issues. In some cases, teams take decisions themselves depending upon the maturity of the team but all these decisions are made in an unstructured manner.

In order to fill this void, this paper proposes a process model and a design of system to assist scrum teams take control of day to day operations. This model and system will be providing recommendations on whether to put engineering effort on features or bugs. This will be done based on parameters like bugs snapshot and various data points related to scrum and team.

2. BACKGROUND ON SCRUM PROCESS

Consider a product scrum team, which is divided in two sub-teams based on their functional areas. One is for taking care of feature development and other is taking care of quality engineering or testing. This scrum team is following an agile process for development where they have scrum of one calendar month where 3 weeks are dedicated entirely to feature development and last one week towards demos, bug fixing and sprint certifications.

Timely builds (nightly and weekly) are submitted to quality engineering team in order to deliver developed features for testing. Testing team files bugs on the features tested with these regular builds. Sprint Submit Build is submitted to quality engineering team at the end of the third week. This build serves as the final feature complete build for that sprint.

As with most of the quality processes, defects in the system under development are measured by a bug process where everydefect in the system is recorded as bug with a severity and priority in a Bug Tracking Tool.

There are two elements to every bug which are,
Priority,giving business perspective and indicates to prioritize the need of a fix over other bugs of lower priority. It varies from P4 to P0 where P4 is highest and P0 is lowest.

Severity, assigned as the extent of technical shortcoming/defect in the feature or system under development. It generally varies from S1 to S4, where S1 indicates a cosmetic defect, S2 indicates minor issue like a malfunctioning feature, S3 is a Major issue and like unexpected fatal error and S4 is a show stopper issue indicating that software will not run.

Generally, Project Manager or Scrum Master actively tracks the impediments and/or dependencies which might block release of various features in sprint daily via scrum meetings. Project Manager/Scrum Master also actively tracks the bug count to measure the quality of the features delivered. The usual glide path based on the bug count is also a measure used to predict the bug count by a date and is a trend curve.

Sometimes scrum team takes bug count as a measure for defects in the system. But bug count of 20 cosmetic bugs is qualitatively

different than a bug count of 20 high priority and high severity bugs. It is just a quantitative measure to track the deliverable's quality status. Other metrics also do not help in estimating and recommending what the impact of bugs is, on sprint delivery and the engineering effort required thereof.

It would be highly desirable for team to have a decision support system which allows them to takeeffective decisions in channelizing their effort on feature or bugs based on day's bug snapshot during scrum meeting. This will make them more self-aligning and self- organizing. And this will also ensure more predictability in final sprint outcome with more predictable shippable features.

3. PROPOSED DECISION ENGINEERING MODEL

Let's first analyze essential elements of bug, which can be represented as a 2-tuple,

Bug B = {Pj, Sk}
Where,

- P is priority of the bug
- j can be an integer value in range of 1 to 5
- S is severity of the bug
- k can be an integer value in range of 1 to 4.

Further, Priority P can be one of the following
P = [P1 | P2 | P3 | P4 | P5]
Where,

P1 = Low priority
P2 = Medium Priority
P3 = High Priority
P4 = Very high priority
P5 = Product ship blocker

Similarly, Severity S can be one of the following
S = [S1 | S2 | S3 | S4]
Where,

S1 = Cosmetic defect in the feature
S2 = Minor failure in the system due to a malfunction in a feature
S3 = Major failure in the system
S4 = Fatal error in the system or a showstopper defect

A bug matrix is a snapshot of all the bugs for the system under development. This can be represented by creating a 2x2 matrix where various priorities and severities are on the axes and values represent the number of bug on a given priority and severity.

Table 1 shows Bug Matrix (as a snapshot on specific day of sprint) =

Table 1: Priority and Severity spread matrix

	S4	S3	S2	S1
P5	$N_{(P5, S4)}$	$N_{(P5, S3)}$	$N_{(P5, S2)}$	$N_{(P5, S1)}$
P4	$N_{(P4, S4)}$	$N_{(P4, S3)}$	$N_{(P4, S2)}$	$N_{(P4, S1)}$
P3	$N_{(P3, S4)}$	$N_{(P3, S3)}$	$N_{(P3, S2)}$	$N_{(P3, S1)}$
P2	$N_{(P2, S4)}$	$N_{(P2, S3)}$	$N_{(P2, S2)}$	$N_{(P2, S1)}$
P1	$N_{(P1, S4)}$	$N_{(P1, S3)}$	$N_{(P1, S2)}$	$N_{(P1, S1)}$

Here N(Pj, Sk) is number of bugs of priority j and severity k.

3.1 Concept

The basic premise of this decision engineering model is based on the fact that engineering effort for building a high quality of software is directly proportional to the number of bugs.

Moreover, priority of a bug, which is driven by business reasons, implies a response time to fix the bug, so a higher priority bug needs faster response time.

Severity which is the extent of defect in the system under development, also indirectly implies engineering effort required to fix the bug, so a higher severity bug generally requires more engineering effort.

- **P is inversely proportional to Response Time required.**
- **S is directly proportional to Engineering Effort required.**

So a scrum team can define their own set of goals for response time and as well as for engineering effort withrespect to priority and severity parameters. For example, scrum team can set a goal of responding to all P5 bugs within 8 hours, all P4 bugs within 24 hours etc. Similarly they can estimate typical time required to fix issues of various severities.

Priority vector = [Rp5, Rp4, Rp3, Rp2, Rp1]
Where R is indicating Response time

Severity vector = [Es4, Es3, Es2, Es1],
Where E is indicating engineering Effort

A scrum team's engineering capacity (EC), which is available engineering effort in a sprint, is dependent on number of factors like,

- Number of available resources in a sprint
- Available hours per day
- Days left in the sprint

EC = (days left in sprint) x (number of resources) x (available hours per day)

3.2Predictive Analyzer

Now in order to automate the decision making process, a predictive analyzer is needed. This analyzer takes inputs, processes it and provides various data points and recommendations for Scrum team to make informed decisions.

Predictive analyzer will take following inputs,
Static data points

- Priority Vector
- Severity Vector
- Engineering Capacity

Dynamic data point

- Bug Matrix

It will process this data, invoke the recommendation engine and come up with following data points and suggestions,

Data points

- Predictive analysis for next few days
- Resource utilization spread for sprint

Suggestions

- Recommendations on putting focus on feature vs. bugs
- Recommendation on bugs to be deferred

As it can be observed, bug matrix is dynamic data point that will change daily. Engineering capacity will continue to go down as the sprint progresses. Although this can be increased by providing support to scrum team by infusing additional resources, but for simplicity, let's keep it as constant for entire sprint. Similarly priority and severity vectors too are constant for a sprint and should be finalized before starting the sprint.

Based on all these data points, predictive analyzer assesses resource requirement based on severity vector which helps in estimating the engineering effort required. Then it utilizes priority vector to estimate how many number of resources will be required in providing a response within the stipulated amount of time.

This data will be hashed out with respect to number of days available in the sprint to come up with detailed resource utilization spread. This resource utilization spread will give a view of number of resources required per day to address bugs.
Here is the process flow,

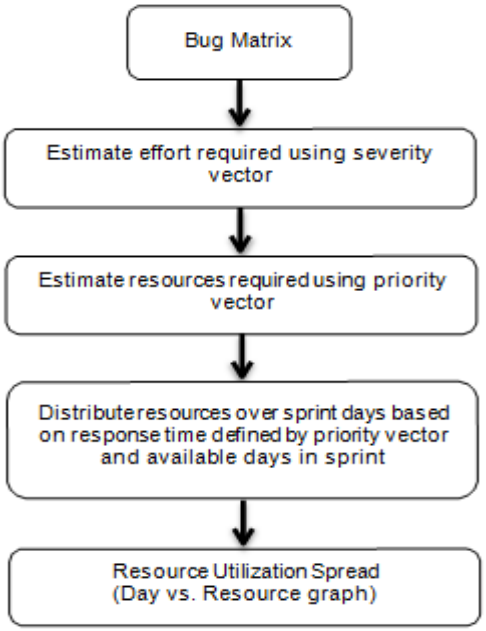

Figure 1: Process flowchart

Now let's take an example to understand this,

Let's assume that Scrum team S is following monthly sprint, which means they have 20 days for product development. S is composed of 5 developers working 8 hours a day. They defined their vectors as follows,

- Priority Vector = {8, 24, 40, 80, 120} in hours
- Severity Vector = {16, 8, 4, 2} in hours

Assume there are two bugs in the system on Day-1. One is B1 (P5, S4) and other is B2 (P4, S3).

So for B1, effort required is 16 hours but it needs be done in 8 hours, so 2 developers will be required. Similarly for B2, effort required is 8 hours and it needs be delivered in three working days, hence one developer can deliver it in next three days.

So resource utilization spread for sprint will look like this,

- Day-1, 2.3 developers are required to work on bug fixing and rest of them should work on feature development
- Day-2, 0.3 developers for bug fixing and rest on features
- Day-3, 0.3 developers for bug fixing and rest on features

It also needs to be ensured that there are sufficient development days left in the sprint while creating the spread. In case, there is insufficient number of days to comply with response time goals, the more resources will be required.

Let's consider a use case where only two days are left in the sprint but Scrum identified a priority 4 bug. Although according to priority vector, team can take up to3 days to provide fixes for bug, but in order to deliver feature in sprint, Scrum team will be required to deliver it in 2 days and hence the resources need to be put accordingly.

3.3 Threshold Limits

Based on this resource utilization spread, decision engineering model can suggest recommendations which can assist scrum team to take informed and quick decision to divert engineering effort accordingly on features and bugs. These recommendations are based on certain threshold limits which are defined by Scrum team.

Lower Threshold Limit (LTL) - This is based on the fact that typical development process requires some amount of active low-effort bug fixing in the system under development which developers tend to address while developing features. This is generally a percentage of available engineering effort in a sprint which is reserved during sprint planning for doing general bug fixing. Typically, teams reserves 5-10% of their effort on bug fixing.

So lower threshold limit is extent of scrum team's percentage utilization reserved for bug fixing, if it crosses this threshold limit, then team is given a recommendation to put more focus on bugs and lesser effort on feature development and below this threshold team is advised to keep delivering features.

Upper Threshold Limit (UTL) - Similarly, it is also a good practice to fix bugs early in the sprint days so as to deliver with good quality. This requires that Scrum team address the quality issues in delivered feature first and then work on next set of features if the resource utilization percentage is exceeding certain limit, called Upper Threshold Limit.

So if the team defines their UTL as 75%, then rather than putting exactly 75% of resources on bug fixing, it will be more prudent to put entire team on bug fixing to attain high quality in already delivered features. So the scrum team will take decision to put

100% of resources on fixing bugs for that day, and then take on next set of features.

3.4 Recommendations Engine

In order to assist scrum team in taking decisions, a recommendation engine will be required. Resource flexibility is another term which implies that whether Scrum team can receive more resources mid sprint or whether scrum team can extend their hours per day to handle spikes in effort required due to bugs.

Recommendation engine provides following effort diversion suggestions:

- Scrum team should continue to focus on features. This implies that the bugs are well contained.
- Scrum team should focus more on bugs.
- Scrum team should focus entirely on bugs. This implies that bugs are not contained and should be handled immediately.

Bug Shifting Process (BSP)

In order to address the case where on a particular day there is a resource requirement exceeding the available resources, then lower priority bugs should be shifted by few days within the sprint. In this way, the ideal response time of lower priority bugs will not be honored but the ideal response time of higher priority bug will be honored.

Bug Deferral Process (BDP)

Bug deferrals are also same as that of Bug Shifting process but it proposes lesser priority bugs to be deferred out of sprint so as to address greater priority bugs in case of effort constraints.
Scrum master should maintain a running list of proposed Bug Deferrals. These bug deferrals can be relooked within the sprint if there is spare bandwidth after all features and bugs are delivered in sprint.

Recommendation process

This process starts with creating a resource utilization spread as mentioned above.

On a particular day,

CASE
Where resource utilization is greater than 100%,
It is checked whether there is more sprint days available in sprint.

- If it is available, then
  - Bugs are moved using the bug shifting process (BSP). After this, the resource utilization spread is recalculated.
- If the sprint days are not available, then
  - It is checked whether there are additional resources available.
- If it is available, then
  - Resource utilization spread is recalculated using one more resources.
    - If additional resources are not available, then
- Bug Deferral Process (BDP) is invoked. After this, the resource utilization spreadisrecalculated.

END CASE

CASE
Where resource utilization is less than 100%,
A recommendation ismade usingavailable parameters.

- If resource utilization is greater than upper threshold limit, then
  - Recommend “focus entirely on bugs”
- If resource utilization is less than lower threshold limit, then
  - Recommend “focus on features”
- If resource utilization is greater than lower threshold limit but lesser than upper threshold limit, then
  - Recommend “focus more of bugs”

END CASE

This process also outputs predictive analysis report for next few days in following manner.

Predictive Analysis

Next 1 day, <Recommendation>, <Average effort>
Next 3 days, <Recommendation>, <Average effort>
Next 5 days, <Recommendation>, <Average effort>
Next 10 days, <Recommendation>, <Average effort>
…
Next 20 days, <Recommendation>, <Average effort>

This whole process is represented here in the flow chart,

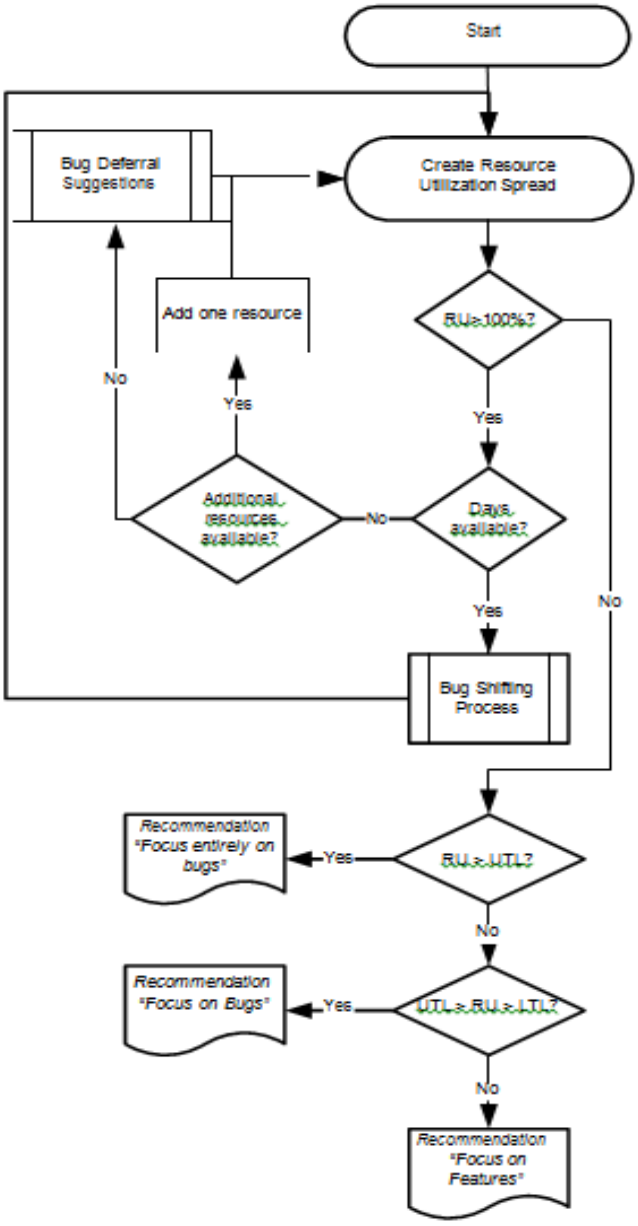

Figure 2: Decision chart

4. GUIDELINES FOR SCRUMS

In order to use this model effectively, scrum team should adhere to following guidelines,

Planning phase

- Define priority and severity vectors
- Define upper and lower threshold limits

Execution phase

- Retrieve Bug Matrix
- Calculate resource utilization spread using Predictive Analyzer
- Take decisions with the help of recommendations

Retrospection phase

- Assess and adjust priority vector in consultation with product owners
- Assess and adjust severity vectors according to team's performance

5. SYSTEM

An interactive system based on this decision engineering model system can be built which can use an adapter to existing Bug Tracking System to fetch the dynamic data regarding bug matrix and provide various reports and recommendations. A detailed design of the system is out of the scope but here is an overview of various layers.

5.1 System Design

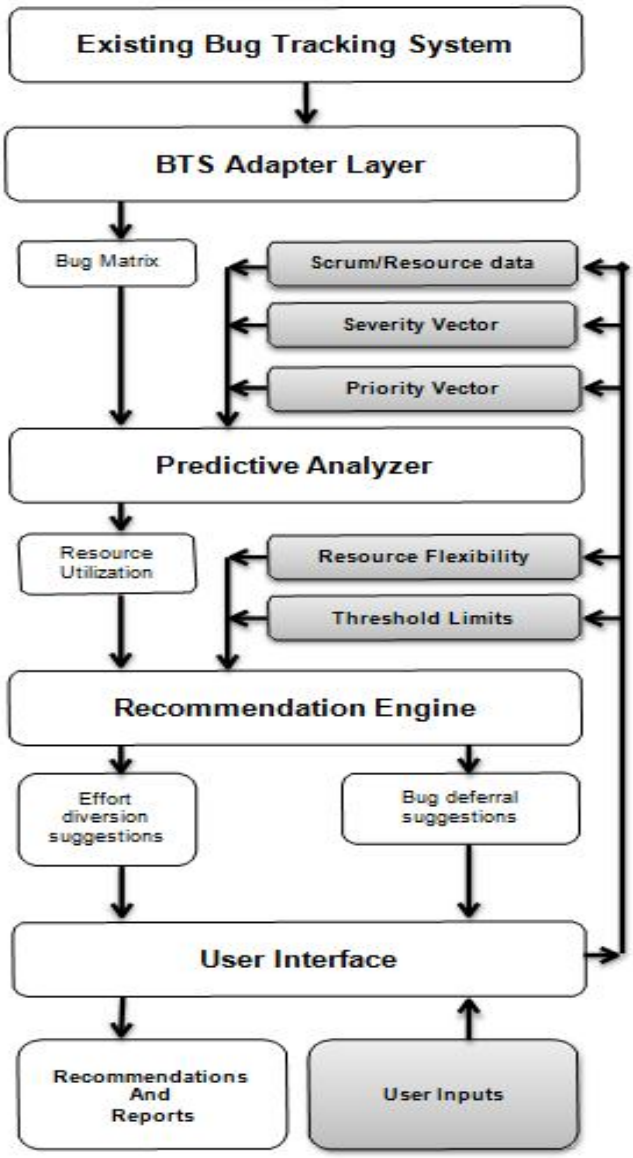

Figure 3: System Design and Components

This system will comprise of following elements:

- Bug tracking system adapter layer (BAL)
- Core System

- Predictive Analyzer
- Recommender layer
- User interaction layer (UIL)

This system plugs in to existing bug tracking system to collect the daily bug data based on priority and severity for the scrum team.

User Interaction Layer or UIL will be responsible for taking data inputs from Scrum team members.This will collect data elementsrelated to scrum or resource data like Number of available resources in a sprint, Available hours per day, Days left in the sprint, resource flexibility and priority & severity vectors etc. It will also collect data elements like threshold limits.

UIL will provide these data elements to core system which will store this data per scrum team. UIL will also handle the job of renderingvarious reports like effort diversion suggestions, recommendations, resource utilization spreads etc.

Core System as explained in section above will be responsible for processing the data elements based on the algorithms discussed in this paper and feed the results to UIL.

6. SIMULATIONS AND ANALYSIS

In order to provide a way to capture and validate the results in absence of real-time system for the proposed concept as defined in previous section, a simulator is developed.

This simulator is representing the real system except the following differences,

It is developed in Microsoft Excel and without any dedicated user interface as suggested in system design
It is not interfaced with real bug defect system so someone in scrum team like scrum master will be required to enter the data manually.

Case Study

A simulation run with a hypothetical scrum team was executed. This team comprised of five developers with four managers; engineering manager, quality manager, program manager and product manager. Program manager was playing the role of scrum master, product manager was product owner and engineering/quality managers were people managers managing engineers on resourcing, effort estimations and feature development.

Assuming this team utilized this model for a sprint of20 day duration and Decision Engineering Model was utilized using the simulator.

Here were the team's parameters,

- Developers = 5
- Working hours per day = 8 hours
- Development Days in Sprint = 20
- Priority Vector = {8, 24, 40, 80, 120}
- Severity Vector = {32, 16, 8, 4}
- LTL = 20%
- UTL = 50%

Day 0

There were no bugs and hence the system recommended team to carry on delivering features.

Day 1

Table 2 shows Bug matrix on Day 1 and is as follows,

Table 2: Day 1 bug spread

		32	16	8	4	
		S4	S3	S2	S1	Total P
8	P5					0
24	P4					0
40	P3			1		1
80	P2		1	1	1	3
160	P1			1		1
	Total S	0	1	3	1	5

As it can be seen, this bug matrix is depicting that team needs to take actions on 6 bugs (1x P3, 3x P2 and 1x P1) OR (1x S3, 3x S2, 1x S1). With help of priority vector, we can see that team's turnaround on the P3 bug is 40 hours, P2 bug is 80 hours and P1 bug is 160 hours. And with the help of severity vector, it is clear that team will take 16 hours to resolve S3 bugs, 8 hours for S2 and 4 hours for S1 bugs.

With the two dimensional data, we can see it will take 44 hours of effort (16h + 24h + 4h) hours based on severity. In terms of resourcing, it will require 0.2 resources per day to finish 1x P3 bug in 40 hours or 5 days, 0.4 resources per day to finish 3x P2 bugs in 80 hours or 10 days and 0.1 resources per day to finish 1x P1 bug in next 19 days. So with this calculation, day to day resource utilization spread will be as follows,

{0.6 for Day-1 to Day-5 }
{0.4 for Day-6 to Day-10}
{0.1 for Day-11 to Day-19}

Here is resource utilization graph,

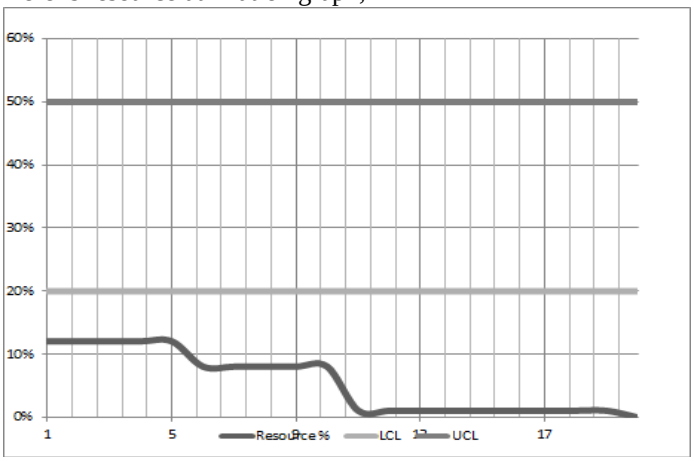

Figure 4: Day 1 Resource utilization graph

Since 0.6 Resource is just 12% of total resource availability and LTL was defined as 20% and UTL 50%. Predictive analysisrecommendation were as follows,

Predictive Analysis		
1 day	12%	Keep delivering features
3 Day average	12%	Keep delivering features
5 day average	12%	Keep delivering features
10 day average	10%	Keep delivering features
20 day average	6%	Keep delivering features

Day 5

Table 3 shows Bug matrix changed due to new bugs in the system and due to few bugs getting addressed by team,

Table 3: Day 5 bug spread

		32	16	8	4	
		S4	S3	S2	S1	Total P
8	P5					0
24	P4					0
40	P3		3			3
80	P2		1	1	1	3
160	P1				1	1
	Total S	0	4	1	2	7

There were 3x P3/S3 bugs introduced which will require turnaround in 5 days and will require significant development effort. With rest of bugs,total effort required is 80 hours.

So with this calculation, day to day resource utilization spread will be as follows,

{1.58 for Day-6 to Day-10 }
{0.38 for Day-11 to Day-15}
{0.03 for Day-16 to Day-20}

Resource utilization graph will be as follows,

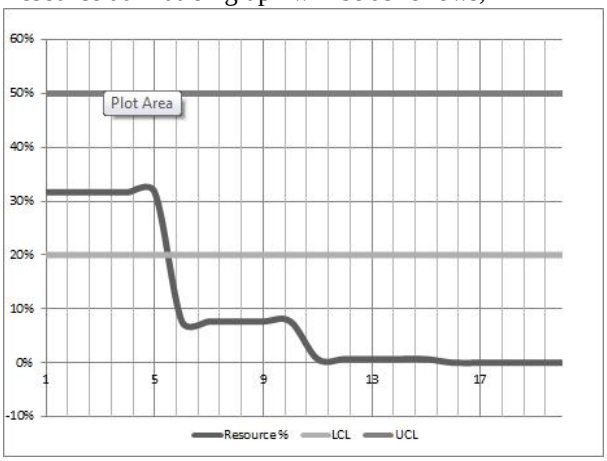

Figure 5: Day 5 Resource utilization graph

Now since resource utilization crossed the LTL, recommendation will be to focus more on bugs as mentioned below,

Predictive Analysis		
1 day	32%	Focus more on Bugs
3 Day average	32%	Focus more on Bugs
5 day average	32%	Focus more on Bugs
10 day average	20%	Keep delivering features

Day 10,

Table 4 shows Bug matrix changed with new bugs and team encountered few P4 bugs,

Table 4: Day 10 bug spread

		32	16	8	4	
		S4	S3	S2	S1	Total P
8	P5					0
24	P4		4			4
40	P3		1	1		2
80	P2			1	1	2
160	P1				2	2
	Total S	0	5	2	3	10

So total effort will be 108 hours and with this calculation, day to day resource utilization spread will be as follows,

{3.52 for Day-11 to Day-13 }
{0.85 for Day-14 to Day-15}
{0.25 for Day-16 to Day-20}

Resource utilization graph will be as follows,

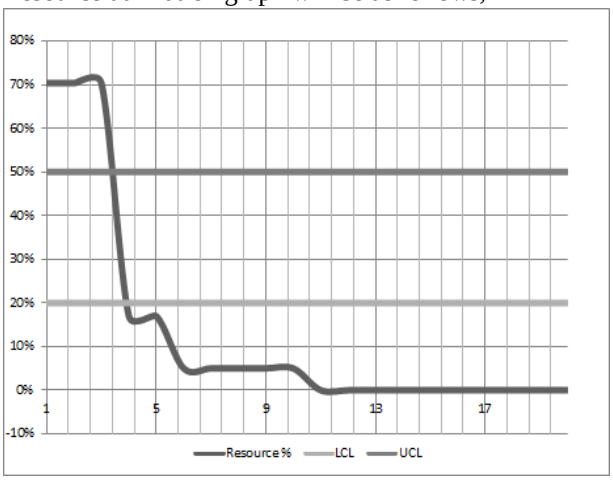

Figure 6: Day 10 Resource utilization graph

Now since resource utilization crossed the UTL also, recommendation will be to focus entirely on bugs as mentioned below,

Predictive Analysis		
1 day	70%	**Focus entirely on Bugs**
3 Day average	70%	**Focus entirely on Bugs**
5 day average	49%	Focus more on Bugs
10 day average	27%	Focus more on Bugs

Day 18

Table 5 shows bug matrix on 18th day and is as follows:

Table 5: Day 18 bug spread

		32	16	8	4	
		S4	S3	S2	S1	Total P
8	P5	1				1
24	P4					0
40	P3			1		1
80	P2				1	1
160	P1				4	4
	Total S	1	0	1	5	7

Total effort will be 60 hours and with this calculation, day to day resource utilization spread will be as follows,

{5.75 for Day-19th}

{1.75 for Day-20th}

Resource utilization graph will be as follows, Resource utilization graph will be as follows,

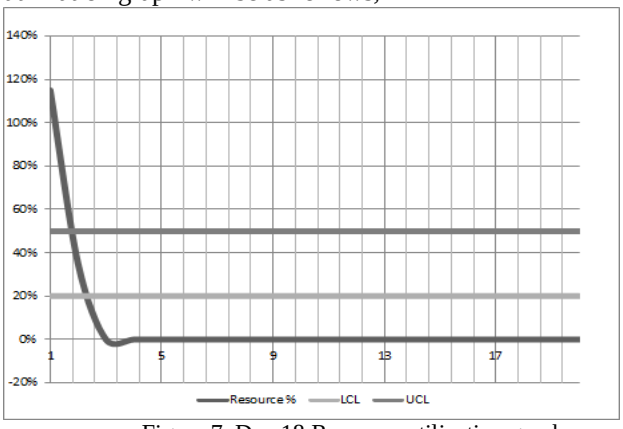

Figure 7: Day 18 Resource utilization graph

As it is evident that in order to deliver the P5 bug which is of S4 severity (32 hours of work), team will require choosing between few options,

- Either put more resources for one day
- Defer work for some of the bugs using the Bug Deferral Process (BDP)

Predictive Analysis		
1 day	115%	**Put more resources**
3 Day average	50%	Focus more on Bugs

7. CONCLUSION

Agile processes require that scrum teams should be self-organizing and self-managing. Proposed decision engineering model and system helps scrum teams in taking informed and calculated decisions in self-sufficient manner during software development process. It is providing them daily recommendations based on existing data points.

The proposed model and system has number of characteristic making it suitable to be used by any organization to utilize it in their software development process.

- Generic – It is a generic model, as it can be used by any organizations utilizing agile methodologies. It can be plugged into their existing software development process where bug tracking system is in place.
- Flexible - This model provides flexibility as there are various parameters which can be customized according to scrum team's preferences.
- Adaptive - It is also an adapting model and provides mechanism to change according to context.

8. FUTURE WORK

This paper proposes a concept for decision engineering model and there are opportunities for further work on detailed implementation of a generic system which can plug into any bug tracking tool to mine the data and present these recommendations and resource utilization spread.

There are further opportunities for detailed study and work on following aspects,

A detailed research to carry out empirical analysis with various scrum teams with this concept to establish effectiveness of this model

A detailed analysis based on data collected from scrum teams to optimize various parameters

- Estimating a team's evolving maturity with a metric based on priority and severity vectors
- Algorithms for optimizing the bug shifting and deferral processes
- Augmenting and extending the scrum systems with proposed model with help of proposed algorithms and set of optimized parameters

REFERENCES

[1] Ken Schwaber and Jeff Sutherland, The Official Scrum Rulebook, Scrum.org

[2] Nils Brede Moe, TorgeirDingsøyr, Tore Dybå Understanding Self-organizing Teams in Agile Software Development, IEEE 19th Australian Conference on Software Engineering, 76-8

[3] Falodiya A., Jain A., Predictive Analysis Driven Decision Engineering Framework for Self-Directing Agile Teams, Research Ideas In Software Engineering and Security (RISES) 2013, India.