UNIVERSITY OF OSLO
Department of Informatics

Investigating Community, Reliability and Usability of CFEngine, Chef and Puppet

Sudhir Pandey

Network and System Administration

Oslo and Akershus University College

May 22, 2012

Investigating Community, Reliability and Usability of CFEngine, Chef and Puppet

Sudhir Pandey

Network and System Administration
Oslo and Akershus University College

May 22, 2012

Abstract

An investigative study on community, reliability and usability of CFEngine, Chef and Puppet is represented in this paper. This research study attempts to quantify software qualities like community, reliability and usability of these products and analyses the result to figure out if any product stands out in any of these qualities. Comprehending software characteristics like community , usability and reliability is complex operation often making it challenging to make a quantifiable measurement on them. Research is made in this paper to explore and make these qualities measurable and quantifiable through the application of different statistical and mathematical model. Product popularity trend, resources available for product, community structure as well as it's field support is studied utilizing different sources like Google, Hackers news and users mailing list. Reliability growth in latest three version of these product is examined by application of Weibull distribution on data obtained from individual bug repository. Finally the usability test is conducted to cover both subjective and objective aspect of user experience on these product to measure each product's usability and study the difference in usability offered by each. This research hopes to pave the way for future research into this area and help people to comprehend community ,reliability and usability of these products.

Acknowledgments

Foremost, I would like to express my sincere appreciation towards my supervisor Ismail Hassan for his support , motivation and encouragement. He has been very supportive and constant source of encouragement through out the project and course. I couldn't have imagined a better advisor and mentor for my masters thesis.

My gratitude to Professor Mark Burgess for his guidance, inspiration and conversation providing much valued inputs on everything regarding the project. His continuous guidance helped me during the time of research and writing of this thesis and I am very proud to be his advisee.

Thanks to Hrek Haugerud and Kyrre M. Begnum for their guidance and their excellent efforts to overcome the challenges faced through out the course. I have learned much from them and the program as a whole.

My sincere thank also goes to Mr Amir Maqbool Ahmed, my fellow classmate Vangelis Tasoulas and my seniors Mr Issac linley, Mr Bishwa Shrestha for taking their time to carry out the usability test which is part of this research. It has certainly been an wonderful learning experience for me.

I am also very grateful to be in a good company of fellow classmates. They have been an excellent companions throughout the course. I would to like thank them for their good will and collaboration.

Last but not least ,my friends and family have always been there for me. Thank you for your support.

Contents

Chapter 1

Introduction

Machines needed to be configured in one way or another in order to make them useful for doing tasks. By configured it basically means installation of of services and application and many more and this is only the most basic operation needed to be performed in a single machine. For example to get a machine connected to the network it needs to be assigned an IP, i.e. it's network card needs to be configured. In another scenario, if a computer is in a network and it needs to play a certain role for example DHCP server, a dhcp service needs to be running on it and for that a set of files is required by this service should be configured.

If it is a single machine and the task doesn't need to be done repeatedly then it is feasible to carry out such things manually but it is not ideal at least in the real world as task described above it needs to be done over and over again. It may be due to machine failure or addition of identical machines in the network. More organization have machines with different operating system and various applications on network thus resulting a heterogeneous environment to manage. One has to spend a huge amount of time doing different adjustment and configuration to make these different system cooperate and achieve an operational network of machines playing their part. If it is the first time of set up these things are usually done manually. But what if it is to be done again and again when new machine gets added to this network and it breaks the whole set up because of different reason i.e. human errors, hardware failure etc. Hence, not only setting up such environment is difficult and time consuming but managing them can be a daunting task if it is carried out manually. More over it is a known fact that IT environment of organization is dynamic in nature and it is impossible to manage it without some kind of automation.

With manual way of managing configuration files of machines we have two problems, the documentation end up in the heads of people and it is not scalable. Also if a same task to is be done multiple times , it is highly prone to human error. Even with most careful approach while following the explanatory notes , it is always easy to miss the details thus resulting a miss configured system. So to avoid such problems scripting was used to automate the task of doing configuration. But the scripting had problem that it was not scalable when the network grew as it needs to be modified every now and then to do new things and was difficult to track as they tend to be scattered all over the place. The scripts thus introduced the new challenge of itself , that was to manage the scripts which were meant to manage the different configuration files. Hence configuration management tools like CFEngine , Puppet and Chef came into existence to get the job done and make life easier.

1.1 Configuration Management Tool

A configuration management (CM) tool is a robot that does work for you, keeping track of the files, packages, services, and other pieces of machines in your environment and keeping them up-to-date for you [1]. These tool works by reading a blueprint document that states how our system should look like and how individual host in network should be configured. What goes inside blue print document is dependency analysis and this is further applied by configuration tool on runtime known as runtime configuration.

Dependency analysis means the task of knowing what things are necessary to get something done. In context of infrastructure architecture it means putting together the layers of services and make a piece of software component working in the whole context. For example, a typical web application might require a running database service and the web service available on a network etc. The runtime configuration relates to the process of taking all the information gathered from the dependency analysis and implementing them in the system. It involves populating correct configuration files , installing softwares , starting process etc. And more importantly all of this should be working even after the system reboot.

By gathering all the information about host configuration in a central repository, it is trivial task to get the exact copy of such host. Just firing up the configuration management tool leads it go through the blue print document and apply minimal changes required for the new hardware and finally get the job done. Same process applies for the disaster recovery as well. For example in case of hardware failure, bringing on the new piece of hardware and letting a configuration management tool do the rest will put the machine back into business without much work from us. In case of software failure next run of configuration management tool is enough to get everything back to the desired state. With the help of these tool greater flexibility is achieved as everything does not need to be carried out from scratch once it is done. These tools are designed to automate much things as possible and reduces the amount of work needed to be done by the human.

Configuration management tool also facilitates documentation. Since we have configuration of each and every thing in a single repository a lot of the work is already done. We can get a list of machines ,tell what jobs they do, and exactly how each of them are configured just by inspecting the configuration files in repository. New hires can have a complete view of our network in it's current state without tracking down every machine owner to find out what exists. Similarly, it is also possible for system administrators to tell the auditors about the current stage of their network at any time instead of scratching the head and telling them about installed packages and configurations. It helps to show what is exactly there within the system, completely eliminating the guess work to find out retired machines.

1.2 Motivation

Configuration management is essential part of system administration. Automation of system administration is a must to handle the deluge; else swarms of system administrators would be needed to handle all these systems [2]. There is a rising demand for configuration management software from large corporation to small business. An infrastructure based in configuration management tool helps to layout a solid foundation which enables companies to achieve faster machine deployment, faster disaster recovery and increased flexibility. Hence it is essential for these organization to have configuration management tool to keep their IT infrastructure up and running 24/7 and achieve agility.

But for implementing the configuration management tool one must first overcome the initial problem of choosing the right tool. Since adoption of configuration management tool is an investment of time and money into future, it is desired that our investment to be fruitful and well paid. We want to be sure that we picked up the right tool that meets the current need and obviously be usable and useful in future no matter what the circumstances are. People want to pick up right technology that can cope up with the changes likely to be made in their IT infrastructure in future. It will be a pity if we have to make a switch on to another tool down the road after two years of usage , additionally at that time it will be virtually impossible and a painful task of switching on to another configuration management tool as everything in the infrastructure will be based on it.

Large variety of configuration management tool are available to this date at various maturity level with different characteristics targeted for different user groups. But it is easier to name the ”big three” in terms of their development stage and install base. Namely they are ”CFEngine , Chef and Puppet”. All of them are mature product and is capable of completely handling an IT environment. Making a choice between one of these is often difficult for a new user and involves a lot of time and effort in trying to evaluate these products following different criteria. A number of things are considered by a common users before being committed to the product and they wonder around in internet to find their answers. Thus the motivation of this thesis is to examine various frontiers like Community, Reliability and Usability of these products providing much information on these topics to users and helping them see the difference if there is any.

1.3 Problem Statements

Number of articles and papers have been published comparing these products that can help the user to adopt a configuration management system [3] [4]. There are also papers that guides the users to make a choice from the products available [5] on basis of various features they provide and the technology they are build upon. Apart from the underlying technology and features about the product, users are always keen on knowing the product's popularity, how big is it's community , can they get their job done using the product , how will they get support in case of trouble. In addition to these queries users often have questions about community support, complains about the products and it's impact on i.e. on scalability , usability or reliability etc. So there are various question that comes on a mind of users when it comes to product selection. They try to find the answer to these kind of question going through forum, discussion sites , benchmarking and testing the products on their own and derive conclusion. But every investigation on these products are adhoc process that ends up in the company documents, kept as private assets which are not accepted as reliable source of information. Thus the target of this thesis is to investigate the community, reliability and usability of CFEngine, Chef and Puppet which can answers majority of the questions, as well as help users to have a insight into those aspects of these products that has never been exposed and analyzed before.

Chapter 2

Background and literature

Today the field of configuration management and automation of infrastructure is a hot topic. It tends to create a heated discussion between the users that have been exposed to the various tools and loyal to it. Also we can see a growing number of new solution to address the problem in the field. For decades this has been the field where the system administrator has been implementing their ad hoc solution to fulfill their need. By inspection of reviews on the web and papers published we can find out that there exist top three open source solution that are widely used and popular. These solution are able to meet the various requirement of infrastructure management as claimed by their sources and users, also there are active company behind the development of these products. Hence we can focus our attention on these products to do the job of infrastructure management. When it comes to the task of picking one from these products ,it becomes tricker and often daunting task for system administrators. It is time consuming process to study their pros and cons and determine which one best meet the need. Often by a quick search on web provides the result that are the outcome of individual evaluation technique and criteria which are some what ad-hoc in nature. Also these results seems to be heavily influenced by individual requirement and experience. So one cannot make decisions based in these results, as the broad view in analyzing these products is not taken into account.

It becomes even more increasingly frustrating when you see contradicting pieces of information from various source about the same product. Previous work done on analyzing these products like comparison of the performance and resource usage [4] or analysis of usability [3] are either too focused in nature or not following a scientific mechanism that accounts all the aspects of the product. Since these products are open source, various things can and need to be in consideration on making a successful evaluation and analysis of these products.

2.1 Open source Assessment Methodologies

The process to keep a list of criteria on which a open source can to be evaluated dates back to 2003 which was started by David Wheeler[6]. Since then a number of work has been done on this area that resulted various methodologies for analyzing the open source software. David revised the list in 2011 which can be found in [7]. At present there exist a lot of methodologies discussed in wikipedia[8]. The wiki also shows comparison chart between the various methods. Later ones were developed for addressing the limitation of the older ones. Among the different methods, the methods like QSOS[9] and OpenBRR [10] seemed to be consider broad range of software aspects organized in hierarchical manner. But these also have limitation and not perfect thus the criteria are constantly revised and updated. At the time of writing the thesis OpenBRR was being revised to get new list of criteria for analyzing the software which is a strong evidence a single given methodology cannot be applied in all the circumstances and methodology must be generic thoroughly revised to meet the need of dynamic IT industry. Therefore a research was conducted as part of The European commission funded project named Qualoss [11] for making a detailed and rigorous assessment methodology comparison between previously discussed [9] [10] and find out their limitation that is shown clearly in paper [12]. The paper [12] aimed to drop the bad points from the previous methods while combining the good points. However the paper contradicts with our way of viewing the criteria list.

Observing the QSOS , it seems to apply criteria of evaluation in 3 levels. Using a precise wording in it's top level , it discusses a list of characteristics in second level which are straight forward and in third level are a set of metrics some what repetitive. But the problem with this model is it is rigid and doesn't permit addition of more metrics in it's top level. OpenBRR method organizes criteria in 2 levels , first level being generic and broad while the second level is clear in what it want as answer, but quite difficult to measure unless it is broken up into further metrics that can be measurable. It is because of this fact many open source software are being evaluated on OSOS methodology [6]. "But can a method be universal" [12] and each and every open source software analyzed by using same number of metric under any kind of circumstances. The article [12] came up with 3 level criteria list on top of which has a list from QSOS but it seriously lack the openness of methodology that user is free to apply. A 3 level criteria list is enough to come of with metrics that can be used for analysis of software product but the top level must be broad as possible to incorporate may sub criteria inside it as felt by the user. A custom model can be constructed we can use the top level criteria from OpenBRR combined other level of methods discussed in [7] that OpenBRR lacks. Second level criteria from both OpenBRR and QSOS can be used to come up with a definite model and further split up each criteria in second level to a measurable metric and precisely describable for all of the tools into account. We strongly believe after assessment of the tools using this model will be present clear picture and supply information regarding almost all the thing that user like to know about these products. Of-course all of the criteria as shown in fig2.3 are generic and can be applied to any open source software in general. It is easy to get started with analysis on more generic term and then focusing finally in the specific particular field where the product is focused.

For a in depth analysis of product it is even important to have a a good understanding of field where the software is used for. A clear understanding of the problem the software is trying to solve is needed. Are there enough functionality in the software that makes a product capable of doing what it takes? Is the architecture of software strong enough to cope up with the future challenge in that particular field etc. All these kind of question are field specific and they need to be addressed for making a strong analysis. Fig 2.2 shows the application specific criteria for analysis

2.2 Literature overview

2.2.1 Leuven university site and paper

An effective framework for evaluation of configuration management tool was presented by a the research group from Leuven university for the large scale system administration conference held in 2010. The framework was then applied on 11 different the configuration management for analyzing these system. The frame work takes 4 main criteria into concern, starting from input specification which is concerned about the configuration language, secondly deployment style discussing how the configuration rules get enforced in the end system. Third criteria is management that focuses on the functionality and scalability etc of the product. The third being support which discusses about the documentation etc for these system. The criteria discussed in the paper is specifically related to configuration management field. The paper is much helpful in capturing the over all picture of what each of the discussed configuration management solution offers and what one should be looking into consideration in order to choose a configuration management product that suits his/her need. It provides almost all the needed information one need to know in order to make a better comparison between the configuration tool and finally make a evaluation of those products that matches the user requirement. The level of details focusing configuration tool in this paper is very granular and elaborative. Different " well thought" aspects of configuration tool is discussed as background and based on those aspect the competitive study is carried out between these tools. Thus it is clearly able to all show those precise things that is needed to make a good configuration tool and tries to explore if those things are the available in tools present in the market. The top 4 characteristics, is further divided in to sub categories to give a clear understanding of scope and area which is explained briefly. The table 2.1 tries to summarize the result for CFEngine , Chef and puppet from their paper [5]

1. **Specification properties**

    - *Specification paradigm* which deals with the type configuration language and the User Interface the product provides. This language is used to specify the user intention as configuration specification. The User interface helps the user to work with the tool implement the intention into language.

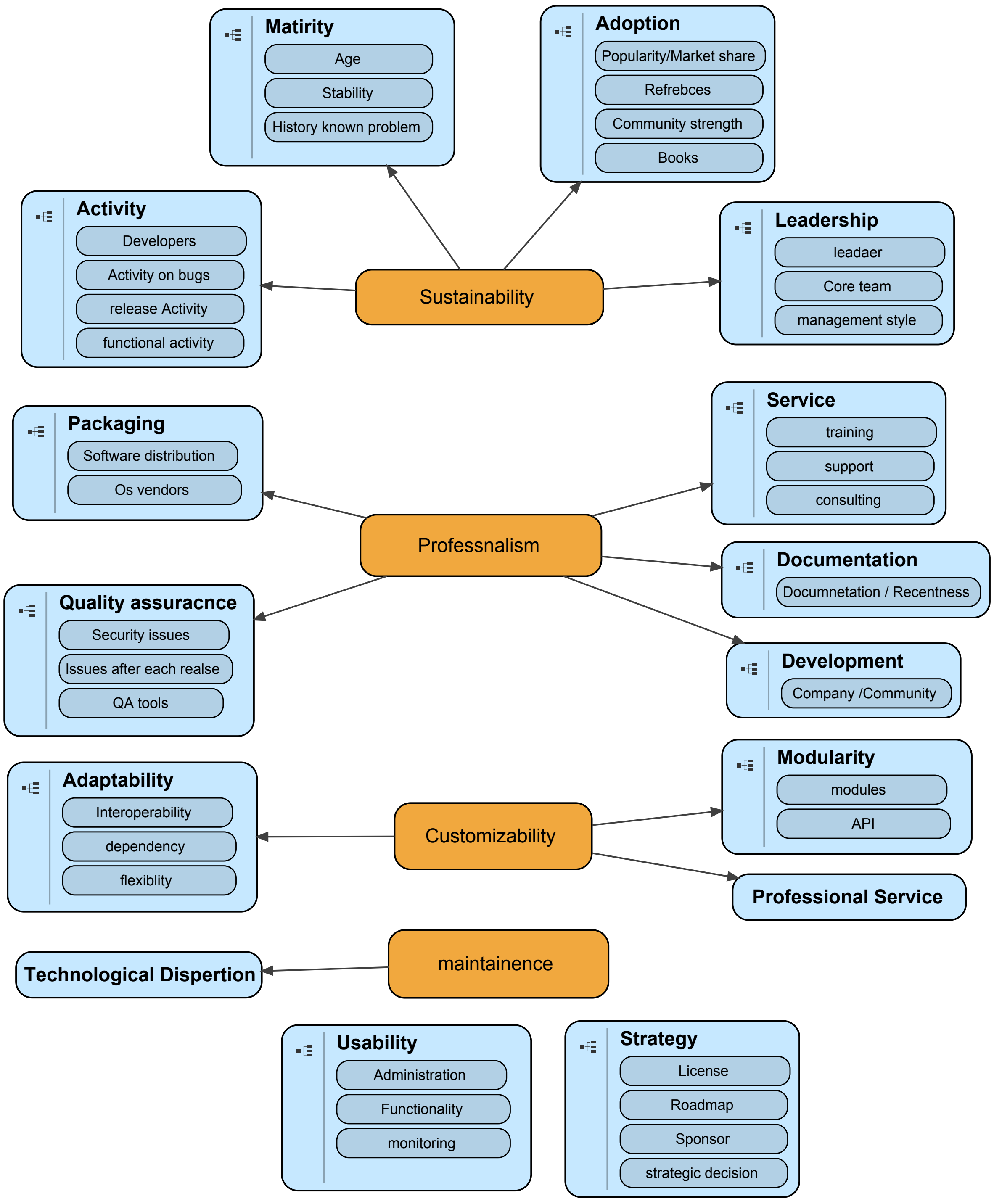


Figure 2.1: Generic Criteria For Product Analysis

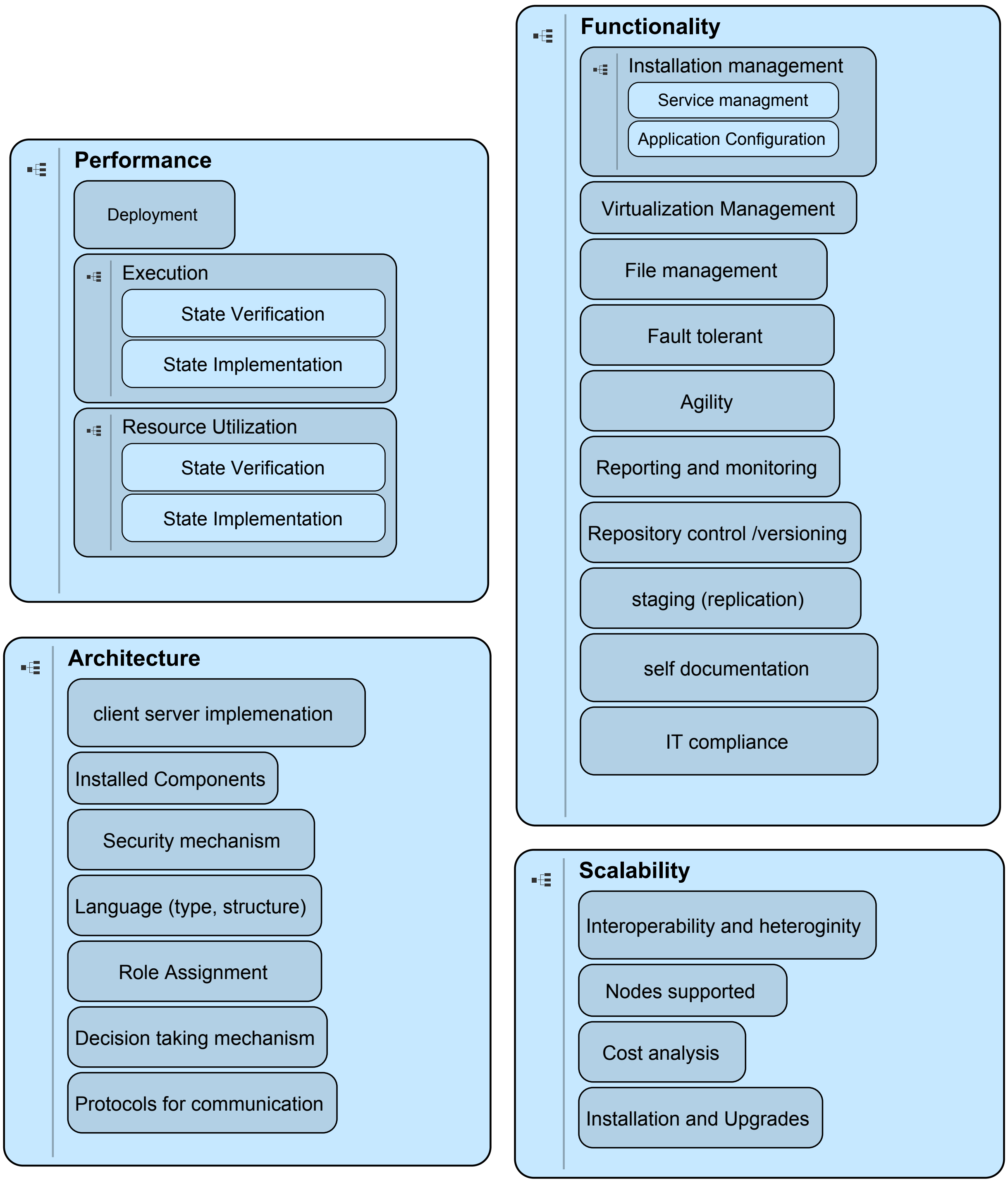


Figure 2.2: Application Criteria For Product Analysis

- *Abstraction Mechanism* is used to provide the details of the abstraction level that configuration tool can provide to implement the user desired state. The paper discusses about 6 abstraction level the tool can provide to deal with the complexity of the infrastructure.
- *Modularization mechanism* explains the different ways the configuration tool provides it's user to make the code reusable. One of the main aspect of using the configuration tool is to avoid repetitive task, those task can be written as configuration steps using the configuration tool's respective language and this code now can be use when ever such task is to be carried out in future.
- *Modeling of relation* tries to views the infrastructure as a system with various components holding different relation ship with each other. It tries to explore configuration tool ability to support these kind of relationship, so when ever some thing is changed the tool can automatically adjust configuration for other. Thus it reduces error and down time in the system and facilitates automation. It categorizes those relation in terms of granularity on basis of instance relation and arity on basis of one to one , one to many and many to many. It is a very typical subject to be studied in the configuration tool which enables the user to determine the tools capability in advance.

2. **Deployment properties**
   - *Scalability* discusses the ability of configuration tool to adapt to the changes with the growth of infrastructure. The tool must be able to provide configuration specification for those large and complex environment.
   - *Workflow* deals with the planning and executing of configuration changes. Keeping this into consideration the authors tries to identify the tools ability to enforce such work flow mechanism that facilitates a smooth transfer of the system state with out any disruption. Smooth transition can be achieved by coordinating the distributive changes and by preserving the state while making change.
   - *Deployment architecture* describes the architecture used by configuration tool to deploying the input specification. The written down configuration specification has to be implemented on each machines , hence configuration tool deploys agent on individual machines that are controlled centrally or act independently. It also discuss how these agents obtain their configuration specification i.e. via push or pull.
   - *Platform support* tend to take the heterogeneity of infrastructure into account and compare the tool on basis of the number if platforms they support. Large number of platform support is always desirable as it plays vital role scalability and interoperability.

3. **Management Properties**

- *Usability* in this paper takes three main things into consideration , Firstly the easiness of the language that enable the user to quickly switch into these tool. Secondly support for testing the specification which lets user understand to see and understand the impact of specification prior to the implementation in production environment. And lastly the monitoring the capability of the tool in itself and possibility of integration with other monitoring tools which enables the user to get information about the current state of the system.
- *Versioning support* helps the user to document and track of their configuration specification. The researchers have tried to identify this feature in all the 11 tools.
- *Specification Documentation* is used to point out the ability of the tool to generate the necessary documentation about the infrastructure from the configuration specification itself.
- *Integration with environment* is used to point out the ability of the configuration tool to consume information from other parts of the infrastructure in it's configuration specification. This enables the users to avoid duplication of information because they need not have to write the information explicitly for the configuration tool. Once such case is consuming the users and roles from the LDAP server.
- *Conflict management* discusses about the possibility of having the conflicting definition on the configuration specification and the ability the configuration tool to deal those conflicts. Different kinds of conflicts like application specific conflicts e.g. cause by binding two application in same port or Modality conflicts e.g. caused by starting and stopping a service in a same machine needed to be detected and acted upon.
- *Workow enforcement* can be regarded as the feature built into these tool that models the workflow while rolling out the configuration specification. Typically a configuration specification passes through various phases i.e. Q&A testing etc and also junior system administrator writing the configuration specification which are to be reviewed by senior administrator the code before being rolled out in production.
- *Access control* is one of the desired feature in configuration tool that allows only the relevant person to write and change the configuration specification. The tool should have authentication and authorization of system administrator in place before making changes and prevent and allow access on configuration specification the based on their credentials.

4. **Support**

- *Available documentation* is used to get a clear picture of the documentation for the tool on various level. The documentation should be brief and offer less barrier to get the novice user started while provide the extensive and elaborative material describing all the aspects of tool for the experienced user.

- *Commercial support* helps to quantify the tool can be trusted and be adopted for use.
- *Community* is important aspect of any configuration tool through which a lot of information can be retrieved to tackle problems and for getting suggestion on tools usage. It needs to be active and lively.
- *Maturity* for pointing out the stability of the configuration tool.

As seen in the table the survey presents a very detailed background theory along with the competitive information different aspects of the configuration tool. The division of the whole task of analysis into 4 categorical views with unambiguous criteria presents a a clear and easy to flow mechanism. The researchers has also introduced interesting aspect like abstraction mechanism and levels , work flow methods and conflict management traits that seemed to be innovative and different from other available methods of analysis. The exploration of abstraction mechanism can leverage one to understand the true potential of the configuration tool. However there are some aspects where the researchers could have done more. For example they have presented a well defined method on which one can study the deployment characteristics but the knowledge about the translating agent and their method of communication is not only sufficient. It is essential to know how the decision taking mechanism is coupled with the implementation of configuration specification, I believe it presents a much clear picture. The agent can be thin doing only implementation or Thick doing all the compilation and implementation of the configuration specification. Hence if the agent does all the heavy lifting it can distributive and scalable as less work is done in server. Knowing only if the agent is push or pull based is only a small part of information what the agent pulls or the server is capable to push can present more information. Also if we take a look at the usability analysis , the researches have ranked the tools difficulty on their own experience and understandability which is a bit unscientific way which might not represent the actual experience of the users. The researchers also have taken scalability into account and have clearly stated they analyzed this metric on real life use case on a single server handling number of clients. But this metric need to be analyzed with carefully performed lab experiment providing same kind of environment i.e. providing same hard ware capability to all the tools.

The survey clearly lacks to explain the methodology that the researchers applied to obtain the result. For example the researchers have given their result of on the community size on purely based on their estimation and their experience. It have been very useful to get some numbers and trends i.e. growth/ decay in the community size. Various methods can be implemented to make a significant research on community size from monitoring the mailing list to monitoring the activities in repository and the user contribution on

		CFEngine	Chef	Puppet
Specification Properties				
Specification paradigm	Language	Declarative	Imperative	Declarative
	User interface	CLI	GUI + CLI	CLI +GUI
Abstraction Mechanism	Grouping mechanism	Classes	Roles	Classes
	Configuration modules	Bundles	Cook Books	modules
	Relation Modeling	one-to-one ,one-to-many and,many-to-many between parameters instances	many-to-many between instances	one -to-many between instances
Deployment Properties				
Scalability	nodes supported	more than10K	1000-10K	unknown
Work flow	Distributed changes	supported	supported	un supported
Deployment Architecture	Translation agent	strongly distributed	central server needed	central server needed
	Distribution mechanism	pull	pull	pull
	Platforms	*BSD, AIX, HP-UX, Linux,Mac OS X, Solaris and Windows	*BSD, Linux, Mac OS X, Solaris and Windows	*BSD, AIX, Linux, Mac OSX, Solaris
Specification Management Properties				
Usability	Tool as a whole	medium	hard	medium
	Specification testing	dry run	multiple environment	multiple environment with dry run
	Monitoring	build in and Integration with other tools	easy integration with Nagios	reports in metrics with in each node and integration with Nagois
Versioning support		svn or git	svn or git	svn or git
Specification documentation		structured comment for generation of documentation	comments on code if structured Rdoc can be used	comments on code can generate reference documentation (limited).
Integration with environment		run time discovery	run time discovery	crun time discovery.
Conflict management		modality conflict	modality conflict	unknown.
Workow enforcement		no	no	no.
Access control		file path based	file path based	roles based inside configuration specification.
Support				
Available documentation		extensive reference documentation on web site	extensive reference documentation on web site	extensive reference documentation on web site.
Commercial support		yes	yes	yes.
Community		large and active	large and active	large and active.
maturity		since 1993	since 2006	since 2009.

Table 2.1: Summarize result of result conducted in[5]

actually using these products. A nice comparison of community size between puppet and chef is presented in this paper [13]. Also the researchers kept maturity into concern but fail to mention the significance of maturity i.e. why was the maturity taken into account at the first place. Knowing how old is the product does not mean any thing to the users but Using maturity for analysis of reliability of the product will be a more interesting topic. We can collect various information from the bug tracker of these tool. A mature product will obviously provide large number of information about it's usage. Hence from it's history a reliability study can be carried to reach the final point of stability. More over the paper only provides a brief discussion on specific characteristics focusing some criteria of the products buts lacks other essential criteria of evaluation like total cost of ownership, flexibility and customizability etc. Over all the paper is nice to get a brief understanding of the field and get a grips of what is going to be analyzed but a a significant researched can be conducted in each mentioned criteria to produce a well explained analysis.

There is already an existing comparison [2] that demonstrate the difference in the language structure of 3 different products highlighting their pro and cons in brief. But it only provides the top level view which helps the community realize their difference in terms of their architecture, language used and the working mechanism. This is helpful to understand these product putting them side by side and see the difference but it lacks the level of detail one wants on these kind of comparison. Neither does it provide a extensive investigation on the languages of these product nor does it discusses the architecture in details. While understanding these products it is essential to know the communication that happens between various components of the software, how different bits and pieces of policy files are linked together and configuration files are generated ,Where are the policy files compiled, what does it take to make a simple policy files, what are the different things needed and where do a user need to put those files. Though it's product specific information it will be much helpful data in analyzing the usability and deployment characteristics of the product. One must need to play around with each individual software to have a clear understanding of the product and get the grammar in grasp in order to document the difference in architecture and language in detail that can be useful for the community.

2.2.2 Comparison by Jarle Bjorgeengen

A different but an organized model of comparison focusing in specific properties was made by Jarle Bjorgeengen in making a comparison between puppet ,CFEngine and Redhat satellite in this paper [14]. This study shows how comparative study between product can be done be able to show the result that is easy to understand and perceive. He has done a awesome job by doing a listing out various aspects that are relevant to configuration management tool and marking them as present and not present in the products like CFEngine, Puppet and Redhat satellite. How ever the comparison fails to show the big picture. By big picture i mean to say a number of things are missing in this comparison, Questions like how well the product is adopted by the users , how good is the support and documentation of these products are still unanswered. For the adopting these product for organizational use case commercial support is absolutely needed and it is good to know on what level, also the other things that can be taken in to account is the leadership of these products which clearly influences the road map of the products and innovation. The ability to contribute in road map and add feature in these products to fulfill some requirement will be a nice addition. It is also helpful to know the frequency of response to a bug and the patches released for these products. The discussion about software reliability seems to be clearly missing in the paper. By doing a historical analyzing of the bugs one can predict the the software reliability by carrying out some probability analysis. This paper is only the comparison between three products focusing in specific properties that is common to all the products and thus shows difference between them.

In 2010 jar le published another paper in which he has implemented a scientific approach for making the resource consumption comparison between the two products Puppet and Chef. He certainly has succeed in displaying the better product in terms of resource consumption making a set of repetitive experiment and finally applying a mathematical model like mean calculation followed by t-test analysis in the sample data collected from the experiment. It is able to show the factual data based on the experiment. The paper is successfully for what it aims to provide but it doesn't describe the tools involved in collection of data from the system. The task like how the resource consumption data was collected is not mentioned at all. The paper shows only the resource utilization by these product on standalone machine, but since these tools are mostly used under client server architecture, it is essential to know about the resource consumption results from both the client side and the server side. More over the due to different architecture of the the products , resource utilization study done under a stand lone machine might not portray the over all resource utilization case. Resource utilization comparison by implementing them to do certain task that resembles IT automation in an organization is clearly shown but the possible limitation is it cannot cover all the scenarios that is exercised by configuration management in organization that possibly consumes a lot of resources which can over turn the result. Therefore the possibility of measuring the performance of these system by bench marking the client server model is wide open to explore. Even though these product supported individual machine configuration where each machine in network has it's configuration files and they manage themselves ,they were most likely to be used for centralized management of configuration figuration where a central server have the all policy files and individual agent in the client machines responsible for implementing the configuration generated from the policy files. The compilation of the policy file can take on either client side as in CFEngine and Chef or in Server side as Puppet. So using this Scalability study can be carried out get knowledge of the nodes (clients) that each server can handle. The test involved benchmarking the server daemons of the respective product.

However there were multiple sources [**?**] [2] , [15] etc. that hinted CFEngine was the most scalable one when it comes to supporting large number of client from a single server. While it seems tempting to test the claims of scalability it equally poses a limited scope of the research and it's usefulness to the community as it kind of known fact and no brainer task as the server side of CFEngine is light weight and used only for serving policy files to the server. But we would like to include these criteria and facts as a part of research and carry out some analysis using it on all the product.

2.2.3 University of Netherlands

Similar study was carried out in university of netherlands by Niek Timmers, Sebastian Carlier which was focused on usability analysis between the CFEngine and Puppet. They have defined the usability in terms if adoptability of the software as the further classification sections like simplicity, reliability scalability etc clearly signifies it. A survey asking questions about the utilization of these software's was send to various system administrator. With hands on experience , result obtained from previous work done and a theoretical analysis of the product architecture they have tried to come up with the answer. The project have done a decent job in aggregating high level factors and coming up with facts available from various sources. The illustration of background theories involved in these chapter and utilizing it to make a sensible argument for the evaluation under different factors like reliability scalability etc is a nice approach. How ever the are significant shortcomings in the work done , we don't think the result obtained from the survey can be relied upon as it is based on the opinion and information from individual. And the information supplied can biased and the statistics can be deliberately changed to give a wrong impression of the product. Similarly the paper has touched the several high level factors that can be used to classify the products but haven't done enough scientific analysis using some model under each section to make comparison. The comparison is adhoc and the reasoning to one product better under some characteristics and vice versa is not sufficient enough. For example the reliability of the system doesn't only depend on the human errors introduced in the configuration policy files , rather the reliability can to be the measure of error prediction coupled with the historical evaluation of failure rates i.e. bugs and analyzing the critical components. Similarly of some mathematical model like Weibull distribution as in [16] on the metrics collected can be used to make a scientific reliability measurement.

2.3 Software overview

Although the products like CFEngine, Puppet and Chef are all seen as configuration management solution , they differ in their approach. They all have unique architecture and decision making process that in-turn applies configuration polices system wide. They all are capable of managing and configuring all the aspects of the system and quite successful in their field compared to other products. They all have taking a different route to address the problem of system administration and automation.

2.3.1 CFEngine

CfEngine is a the outcome of research conducted by prof Mark Burgess, mainly exploring the topic of configuration management. Mark Burgess is the Earth's first Professor of System Administrator [17] and winner of SAGE Award 2010 [17]. CFEngine is now available as both free project (core) which have the core ability to configuration management and a commercial product with additional features added to the core to meet various organizational need. CFEngine was the first generic tool of its type for providing context based implementation of intended state, and has been around since 1993. Since it's birth CFEngine has transformed gradually into a mature tool rolling out two major versions CFEngine 2 and CFEngine 3. CFEngine 2 was first solution of its kinds and widely accepted and implemented solution through out the industry. Version 3 was rolled out in 2001 as complete rewrite to address the experienced design demerit of CFEngine2. CFEngine 3 is based on the promise theory which was also developed by mark burgess. CFEngine uses a declarative language to write rules for achieving the intended state. The rules are termed as promises and according to the promises being kept determines the final state of the system. With CFEngine3 users were able to build modular reusable libraries of common configuration tasks directly in the configuration language, and a cleaner separation of data types in the language giving less room for misunderstandings. With the introduction of CFEngine 3 the company was formed to offer the enterprise version of the CFEngine to address the reporting needs of organization and to help organization achieve various IT compliance standards like ITIL etc. The commercial version also comes with a set of pre build configuration policies that are often used for carrying common tasks.

Design

CFEngine is strongly distributed when it comes to deployment. Every node can run on it's own and have their configuration files for achieving the target state. However it is possible to maintain centralized configuration repository and make each and every machine to take configuration files from the central server to achieve their target state. It is made possible by the used of daemon called cfserverd. When CFEngine is installed each machine has a set of daemon running in them that carries of various task like implementing the configuration , communicating with the central server and making sure the the compliance is checked at equal interval of time. The dameons can talk to each other when needed which Mark Burgess terms as orchestration. Due to these daemon a single machine is self sufficient making it's architecture distributive. Also each host utilizes one of these daemon to communicate with each other in the CFEngine world.

- **cf-promise**: It is a binary that is to be utilized to validate a policy files manually before making an actual execution. It is helpful in debugging errors in policy file and making sure that the rules(promises) in policy file does what it is suppose to do.

- **cf-agent**: It is the program responsible for executing the codes in the policy files and implementing in them in the system such that the system achieves it's target state. This daemon is ran periodically in some interval (usually 5 min) so keep the system compliant.
- **cf-server**: This daemon is utilized by CFEngine to share share configuration files and also to receive request from remote machine to execute it's local copy of configuration policy files.
- **cf-exec**: It has a similar functionality of crond in the system and is utilized for carrying out scheduling task. The most obvious used case if this daemon is it is utilized to run cf-agent at every 5 minutes so that the policy files get executed.
- **cf-runagent**: This daemon is utilized to make request to the remote machine's cf-server for running it's cf-agent. Utilizing this it is possible to create a push scenario but it all work get completed under voluntary cooperation between host. A host cannot make a remote host run it's cf-agent , it can just request it to do so and then the remote host receiving the request can carry out the requested activity as per it's will.
- **cf-report**: It is the binary that is responsible for keep in the output summaries of each run of cf-agent that will used for producing the reports.
- **cf-know**: CFEngine uses this binary for keep track of 'who did for what and when as mentioned' in the policy files. The language gives you the possibility of mentioning all the possible details about the change and cf-know constitutes the knowledge based documentation from these sources that will be useful for future changes to come.

Hence each individual host equipped with these daemons now is capable of sharing and reeving configuration files , which makes it possible for using CFEngine to be used in client server model. But here the central server only shares / transfers it's file to the client and all the execution of policy files and implementation takes place in the client. Here the server acts like file server. File transfer takes as the result of request from the client. Hence it is the pull model under which the CFEngine works. Clients maintain their local cache of the configuration files and if any new changes is detected they make a request to the sever to get the new ones. Client them selves are responsible for decision making by executing the policy files. This kind of model suitable for scalability as processing power get distributed across the clients and increases the reliability as it doesn't create a single point of failure when central server dies.

Vocabulary

CFEngine is provides a declarative language to express the intended state of the system based on the context. Language of version 3 is flexible enough to declare the expression of intent and the way of implementing it. With the introduction of bundles2.3.1 and bodies2.3.1 it has provision of creating on reusable module for carrying out common repetitive task. So a configuration code can be written for one task breaking the whole task into down into modules and now set of modules from previous task can be used for carrying out new configuration task. It can be a bit overwhelming when one needs to to define the structure of implementation logic in a reasonable way for facilitating reusability from the very first, but the basic building blocks and examples of structured CFEngine code that can be reused, is widely available in open source repository like Github. It is also provided a a part of enterprise package thus by saving time and effort of users.

- **Promise** Every thing is CFEngine 3 can be expressed as promise. They are the basic and only form of expression in CFEngine. Simply laid out a promise is configuration rule. CFEngine language has various promise type that enables users to perform different task. For example if user wants to work with file he can used file type promise, if it is firing a shell command he can use command type promise. A promise have promiser (the abstract object making the promise) , promise e (the abstract object to whom the promise is made) and set of association called the body of promise. Combining all the above with promise type once can understand the intent of the promise i.e. what a promise is suppose to do. Lets have a look at sample policy file which created a file named text.txt and updates it's touch time.

```
files:             # this is the promise type (e.g. files,
                          # processes, commands, etc.)

"/tmp/test.txt"           # this is the promiser, the part of
                           # the system that will be affected by
                           # the promise.

create => "true",                # This is the promise BODY.
touch => "true;                  #The promise body details and
                                #constrains the nature of the
                                #the promise. It consists of
                                     #attributes which have values.
```

- **Bundle and body** A Bundle is a group of promises. A promise is capable of doing a single elementary task i.e., switching the service on or off , editing a file etc. Configuring a single application to work can take one or more of these kind elementary task for e.g. To make a web service to be available in a system. The package apache needs to be installed, it should be started and if ssl need to be enabled then it's configuration file need to be edited. Hence Bundle allows a grouping of the promises that are used for completing a the whole task. If designed strategically elementary task that are repetitive can be converted into promise and placed in the same bundle such that the same bundle can be used over and over to carrying out task that involved the elementary jobs represented by the promise. As seen above the body of promise contains a set of association. The right side of promise can contain a value or a function that does the most minute job as possible like inserting in a file, copying from a server etc. This job can be abstracted in a function and those function can be utilized many task while carrying out those task. CFEngine Community Open Promise-Body Library(COPBL) CFEngine Standard library contains a collection of those functions that can be reused many time while writing a policy file. It is an interface layer that brings industry-wide standardization of CFEngine configuration scripting and hides awkward technical details [18]. An example of body declaration inside the standard library file is presented below used for copying files from a remote machine is presented below.

  ```
  body copy_from remote_cp(from,server)
  {
  servers     => { "$(server)" };
  source      => "$(from)";
  compare     => "mtime";
  }
  ```

  This body can now be utilized in promise when ever we need to copy files from a remote server. The only thing we need to do now is to include the file containing the declaration of this body file , in this cased we need to include the standard library file. An example of making use of the above body is shown below

```
body common control
{
bundlesequence  => { "my_file_copy" };
inputs => { "cfengine_stdlib.cf" };
}

bundle agent my_file_copy
{
files:
  "/home/user/tmp/test_dir/file"
   copy_from => copy_from remote_cp("/var/cfengine/testdir/file","serverhost"),
}
```

- **Classes** CFEngine uses classes to determine the context of implication of policy rules simply answers When and where are promises made, hence class is basically a context in CFEngine. A policy file contains declaration of many promises. Each promises is declared under certain context making is applicable under that context only. When a agent executes a policy files it matches the classes in it ,with the environment variables like OS name, ip address of the system termed as hard classes. If a context match occurs then only it applies that promise in the machine. A general class like 'debian' represents a set of machines having Debian as OS. But a Specific class like '10_0_0_4' denotes only one machine. Thus the implication of rule in host solely depends upon classes, simply said if host falls under the declared class the rule is applied other wise not. Hence care must be given while choosing the class other wise it might trigger unwanted state implementation in other host of the system. Each run of cf-agent t discovers and classifies properties of the environment or context in which it runs. The properties discovered is termed as hard classes. It is also possible to us combine these classes by using different logical operation and define a class out of it. These class are termed as soft classes. Soft classes also can be formed by utilization of special function which are well documented in CFEngine reference guide. Here are some examples of hard classes found on a platform by agent run.

```
any Saturday Hr11 Min26 Min25_30 Q2 Hr11_Q2 Day2
     August Yr2008 linux atlas 64_bit linux_2_6_22_18_0_2_default x86_64
     linux_x86_64 linux_x86_64_2_6_22_18_0_2_default
     linux_x86_64_2_6_22_18_0_2_default__1_SMP_2008_06_09_13_53_20__0200
     compiled_on_linux_gnu net_iface_lo
```

And here is an example that shows the utilization of special function for declaring a soft class and utilizing it in promises declaration.

```
body common control
{
bundlesequence  => { "example" };
}

bundle agent example
{
vars:
 "binary" string => "/bin/ls";

classes:
  "isexecutable" expression => isexecutable("$(binary)");;

commands:
  isexecutable::
           "$(binary)";
}
```

- **Policy**: They are the name given to the files that contain CFEngine promises and bundles and body. Hence in CFEngine language the policy files are the one that contain configuration rules are usually of extension.cf.

Working mechanism

The working mechanism of CFEngine deployed as client server architecture is described here. Every thing except client server communicating mechanism hold true if CFEngine is deployed in a single host. As described above each host installed with CFEngine comes with a set of daemons , if a machine is to be made server then configuration files under /var/cfengine/masterfiles will be fetched by other CFEngine clients. Hence a set of files can be checkout out for any repository source to the /var/cfengine/masterfiles directory. The server achieve it's target state by copying it's file into /var/cfengine/inputs directory and then executing these files by cf-agent daemon. While executing cf-agent it checks for difference in the files under Inputs directory and master directory, if there is any difference the most recent version of files are copied in the inputs directory and then executed.

For client to get the policies file , they need to boot strap with server utilizing servers ip. Boot strap process triggers the cf-agent to execute the embedded failsafe policy which will make a request to the cf-serverd of the server machine to serve the policy files under it's master directory. The communication is facilitated by the key exchange between server and client and the trust is established, this trust will now be utilized when ever the client agent needs to connect to the server. Now the server's policy files are now kept under /var/cfengine/inputs directory and finally get executed. On every run of cf-agent it check the difference of files and contents between it's local copy and the remote copy in server. Incase of difference most recent version is downloaded and again kept in /var/cfengine/inputs then executed to make the system achieve it's target state. If it cannot get the files from the server then local copy is utilized to achieve compliance. This makes it quite robust and fault tolerance as the problem in network doesn't cause the deviation of client from desired state. Of course if there will be some deviation if the change was in server and it was not transferred to client due to some network error but that will be fixed automatically when the network error is fixed and client is able to communicate with the server. This makes CFEngine a very adaptive and flexible product.

2.3.2 Puppet

Puppet was introduced in 2003 by reductive labs. It was basically written by Luke kanies to achieve productivity and simplicity on the field of infrastructure automation. It aims to make the system administrator life easier by hiding the implementation detail of configuration with a introduction of it's declarative language that focuses in getting things done through it's operating system abstraction layer (OSAL) [**?**]. The tool was written out as an alternative to CFEngine 2 and have tried to focus it's usability targeting what user wants to do with this tool rather than focusing on how user should use this tool keeping operating system details in mind. The basic philosophy behind the tool is a abstraction layer is required step in providing the best automation tool,instead of coming up with a new way of handling each OSs messy details. Puppet facilitates code reusability and modularization through the use of class inheritance and it's abstraction layer. When a configuration rule is written in puppets language some of common generic task can be divided into class and other class doing specific job can inherit those generic classes and If a rule will be written for making configuration in one operating system and ideally that rule can be used in any other environment to do same configuration as all how to implement that configuration details is handled by the abstraction layer. It has similar concept like java VM or .net framework which makes it's possible to run their code in any environment as long as the frame work is installed. Though it offers simplicity there might be some cases where the abstraction of some implementation is may not be defined in abstraction layer, This kind of situation can limit the product usability however you can get around it by writing your own implementation logic termed as driver in the ruby code as puppet itself is written in ruby.

Design

Puppet is built with a focus on client/server configuration within an infrastructure. All the configuration rules written in puppets Declarative language reside in the server termed as Puppet Master. Here the puppet master is central point , which does most of the task like analyzing the configuration files and supplying client specific configuration only to the client by puling the necessary things from it's manifests and compiling them in catalogues. Thus catalogs are just data in xml format, not an executable code. All the clients must be connected and trusted by the Puppet master in order to get the catalogs it need. The clients makes request to the server to get it's catalog i.e. set of data that shows the end intended state of it's resources , so configuration implementation mechanism is pull based. The puppet agent in client is responsible for only implementing the things specified in the catalogue once it is obtained from server in case of deviation from server. Once done, the client can send a report back to the server indicating if anything needed to change. The puppet master can also notify it's client when the configuration files are changed so that it will trigger request from the client to get the catalog.

This design has some advantage as everything is controlled centrally i.e. central point of management and client only gets the data file that tells what it's resource state should look like. In case of any client is compromised the whole infract structure is not vulnerable. Also it possible to collect the reports about all the clients centrally as all the client report back to the server. How ever the demerit of this design is when the puppet master is down the client cannot monitor itself to stay in a configured state which might lead the client to undesirable state. This creates a single point of failure commonly seen client/server architectures. Also can present limitation on number of clients that can be managed by central server because when number of client grows the resource utilization of server increases linearly.

Vocabulary

Puppet provides a declarative language whose main purpose is to get the intended state implemented in the system with out caring the implementation details. Hence it offers simplicity and productivity in writing the configuration files without tangling much the details of platform in which the rule is implemented. This is all made possible with a component called resource provider in puppet eco-system. Resource provider can be viewed as backend driver that implement support for a specific implementation of a given resource type by taking high level parameter while making a resource declaration in the configuration language. But what actually is resource.

- **Resource** : In puppet's language resource is objects specified that helps to manage a single the a component of the infrastructure. Each Resource typically have a type, a name and a set of attributes that needed to be implemented. Resource can be thought to be the way of expressing the most elementary form of our intent. Our intent can be writing to a file , executing a command or making a schedule in cron tab etc. The example below shows the our intent of setting a permission on file. so the resource type here is file and it's name is "/etc/passwd" and it's attributes is the associative name value pair parameters follows it's name that is applicable to the type file. There are different kinds of resource type that lets us do wide variety of task like exec for executing commands, cron for scheduling.

```
file { "/etc/passwd":
 owner => root,
    group => root,
    mode  => 644
  }
```

The Resource provider must have some kind of knowledge "logic" that helps it to implement the rules i.e. in different kind of low level context. Ideally speaking the resource provider is like a translator that translates the high level puppet DSL language to platform specific implementation details and get it implemented. It should also be kept in mind that not all resource types have or need providers, but any resource type concerned about portability will likely need them. Hence one need to create the resource provider in ruby language for resource that doesn't have any resource provider or is unsatisfied with the provider way of implementing things. It is often useful and easy to use if the resource provider is found for the resource but when one needs to make resource provider then it is painful and time consuming task. The path taken by reductive lab is that whenever one needs a new resource type he/she will create it and share it so that the community can make use of resource provider. Also for the resource provider to work it needs to know the type of resource and the node's facts.

- **Class**:Class is a collection of resources. It is one of the way that puppet achieves encapsulation. Class are just a name given to related set of resources. Basic purpose of class is to hide out implementation detail and use only the name of the class when in comes to usage. Class can be inheritable so that the most generic tasks can also be grouped in single class and inherited in another class for code re-usability. Shown below is a basic declaration of class including two resources for installation of apache and make it running.

  ```
  class apache {
  package { apache: install => latest }
  service { apache: running => true, requires => package[apache] }
  }
  ```

- **Node**: Node in puppet resembles a machine and is a special kind of class. Node is another abstraction mechanism mechanism that enables puppet to show the intended state on particular machines in uncluttered way. An example of node declaration is shown below where class apache is applied to host named test. This way the intended state of each node is clearly stated masking away underlying classes that takes care of the different levels of implementation. The implemented classes can be nested many level separating the granularity of the task they perform

  ```
  node test {
  include apache
  }
  ```

- **Modules**: A Puppet module is a collection of resources, classes, files, definitions and templates. The difference between modules and classes is that class can be global i.e can contains class definition for doing various task for configuring various application, while module must only contain definition doing a particular job i.e. configuring one application. For example When we need to set up a system with apache service then we need to carry out multiple task like installing apache package, write some thing in a apache configuration file and finally ensure the service is on. Modules can generally be thought of as a configuration containing each of our three core requirements, installation, configuration and monitoring. Big the task is huge module often can be broken down into class and sub class but all these classes are coordinated to get a single big task completed. Puppet looks in the init.pp file under manifest directory in each and every modules directory under /etc/puppet/modules. Module provides a way of code re-distribution.

- **Facts**: In puppet the context is provided by a separate component called fracter. Fracter are installed in the clients and they supply the environment specific things like OS,version,hw-arch, interface ip-adresses, mac-addresses and so on called 'facts'. It is also possible to declared user defined facts in the client. These facts are utilized by the server to pick out configuration rules related to the hosts. In the example shown below we are using the fact name operating system to determine the location for placing the file. Of course it goes beyond the principle of puppet when you use factes for making decision based on the platform but there are some cases where we need to make switching that doesn't depend in the platform details

```
file {
    name => $operatingsystem ? {
        debian => "/etc/php5/apache2/php.ini",
        default => "/etc/php.ini",
    }
    owner => root,
    group => root,
    mode => 644,
    source => "puppet://php/php.ini"
}
```

- **Manifest**: They are the name given to the file containing the configuration rules. So each and every file with puppet's declarative code for configuring the infrastructure are puppet manifest and are of extension .pp

Working Mechanism

The configuration rules manifest2.3.2 contains the a set of resources2.3.2 and classes that specifies what need ed to be configured where. All the manifest

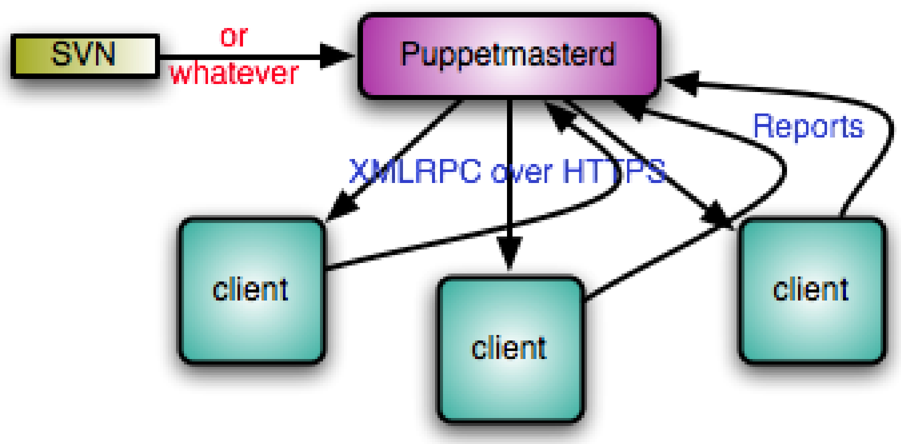


Figure 2.3: Puppet infrastructure design

reside in the puppet master. The client with puppet agent is responsible for implementing the states as supplied from the puppet master. Hence in puppet master there is one daemon running listening to request of puppet client , the puppet client sends request with it's node2.3.2 name a list of facts 2.3.2. The facts in client are collected by fracter which is independent ruby library that collects host specific details like IP address, Operating system name etc.

With the help of the node name and facts supplied from the client , puppet master now classifies the client and decide which portions of the configuration rues is applicable to the client. While doing the compilation and generating the catalog for client puppet master goes through different manifest. The first manifest it looks into is site.pp under /etc/puppet/manifest to get then it goes through nodes.pp under same directory to get host specific configuration. Now it goes through each and every modules init.pp to get all the classes and under modules defined. After fetching every things from manifest file to server is now able to compile them into a set of things that need to be done in xml file called catalog and send to the client. Now the client receiving the catalog run a query to find out it's status and if there is any deviation of it's state from the catalog it corrects it's stage to that specific things mentioned in the catalog. The report about the specific task i.e. the change made is then supplied in to the server.

2.3.3 Chef

Chef was introduced in 2009 [2] by Ops code and regarded as young compared to the other two products. Though young it cannot be counted out as it has been gaining grounds and has established itself to be a top player in short span of time. Chef was designed keeping ”infrastructure as code ” in mind such that it exposes obvious flexibility and simplicity to the user for managing their infrastructure. Chef motto is similar to that of perl language i.e. ”There's more than one way to do it”. There for Chef provides a Domain Specific language(DSL) along with the ruby code can be written which empowers the user to do much things without cornering themselves. This also have facilitated the Chef users to achieve flexibility as they can always achieve things in one way or another which in turn can assist interoperability. No longer is infrastructure is thought separate concept and a new language need to be learned for managing them. This has driven chef successfully in the DevOPs area where a developer can write code for there application and at the same time take care of infrastructure.

Another simplicity Chef introduces is in order of implementation of rules as specified in configuration file. Chef has a deterministic behavior when it comes to applying a configuration rule specifying a component in infrastructure. This enables the user to visualize what gets executed and applied first and ultimately help them to debug for errors and apply their existing knowledge about programming in the infrastructure code. Chef allows a list of resources to be declared in their recipes and execute in the same order as they appear. Chef also uses techniques referred by various terminology to enables user to make a modular implementation of the configuration rules.

Design

One of the core principle of Chef is "thick client and thin server" but the architecture is essentially client server. Chef does much of it's task like compile and execution in client. Chef server is central machine where all our configuration resides on. Chef Server is responsible for transferring all the files as asked by the client and it also stores the state of each node i.e. client machines. This model like CFEngine is scalable because the processing task is distributed throughout the infrastructure. Client accesses these data stored in the server with the help of Rest API. Chef server is not just the file transferring server but has lots of other services. These services are need to make communication with the client devices , letting users interact with server, to store data about the configuration as well as states of it's nodes and search service that allows to query for data stored by chef related to infrastructure. Below are the services that are running in Chef server to make the whole architecture come alive.

- **API Service** It is this service that each client machine interact with to manage the node configuration. Machines with chef-client binary are machines , this binary make communication with Chef server i.e. chef-server process with the help of REST API calls. User also utilizing this service while interacting with Chef-server through the used commands like Knife or Open Source Chef Server Management Console.

- **Open Source Chef Server Management Console**: It is the web UI exposed by the Chef server reachable from port 4040. It provides the user the opportunity to carry out all their infrastructure management utilizing the web. This service is user password based authentication and interacts with the Chef Server making REST API calls. Hence essentially it's API Service client that displays configuration data in web page and lets user to add/modify those data with the help of Web GUI.

- **AMQP Server**: Chef utilizes this component to keep the index of data stored in it's backend because chef provides a powerful full text search about information of infrastructure and applications. Chef uses RabbitMQ as an AMQP server. RabbitMQ provides a queuing service which stores requests for updates to the search index. This allows the API server to handle load spikes while maintaining an even load on Solr. Every time data is stored in the database, server sends a message and the data payload to the queue, and the indexer picks it up.

- **Search Indexer**:The search indexer of Chef is chef-solr-indexer listens to AMQP for messages about what to index, and then passes the results to chef-solr. chef-solr is a wrapper around a Apache Solr. This enables Chef users to make a full-text search to query information about the infrastructure and applications Searches are built by the Chef Server, and allow users to query arbitrary data about your infrastructure.

- **Data Store**: Chef Server uses Couchdb as it's backend to store all the data about Infrastructure. The data are in json format and it holds information on Nodes, Roles and Data Bags.

Hence the setting up a chef server is a complex task and many things need to be installed and ensured that each of them are running. While it is complex it provides a powerful features like querying the data , using the web UI to manage infrastructure and through exposing it's data through REST API any application can consume it which ensures interoperability in the complex environment. As described above the Clients only have chef-client installed in them and interacts with server via REST authenticates via Signed Header Authentication, down load, compiles and executes configuration rules called Cookbooks in Chef world.

Vocabulary

Chef configuration is written in pure ruby which is called as chef domain specify language. So ideally every thing about ruby code can be implemented in the chef configuration file. Thus chef language is imperative by nature. The goal of chef was to keep the language very simple by focusing on idempotent resource declaration and at the same time offering the flexibility of 3GL language [15]. Managing the infrastructure using ruby will be inspiring for a rubists but it there is still a bit of learning curve for new people. Chef creators also believes that there is always a learning curve for newbies but the main difference is when some one hit limitation, they can always find innovative ways to solve their problem. Chef allows users to write resources in ruby. Resource can be any thing like file a package etc that need to be configured. These resources can be listed in an order called recipe and chef executes these resource in the following their written order. Hence chef have a implicit ordering mechanism in which it implements things. So if like in conventional programming the essential things are declared first then followed by the things depending on it. So with this approach chef manages to achieve convergence in a single run. Chef uses couple of interesting terminology that are used to reference a configuration rule or an component of infrastructure. This also be viewed as the naming convention is done in Chef way.

- **Resource**: Resource is an component in the infrastructure and it can be any thing like file to be written, package to be monitored ,or services that needs to be either switched on or off. A Resource must have a type and a name and a set of actions that specifies the target state of the resource. Basically Resource are Ruby objects with the code behind them to configure your system. Behind each resource are one or more providers that tell Chef how to execute the actions you require against a certain type of system. It has similar concept to the puppet that provides a cross platform abstraction of a system's configuration, the only flexibility is that you can inject ruby code like if else etc while making resource declaration. An example of resource declaration is shown below which has pretty clear syntax to understand what the intention is.

```
package apache2 do
    case node[platform]
  when centos,redhat,fedora,suse
    package_name httpd
  when debian,ubuntu
    package_name apache2
  end
  action :install
end
```

- **Recipe**: A Recipe is the collection of Resources in an ordered list. Recipes are written in normal Ruby extended by the Chef resource domain-specific language. It can be termed as resource runlist. The Resources are written in systematic order in a recipe and hence they get executed (applied to node) in the similar order as they appear. This shows the deterministic order of the Chef which is implicit order mechanism defined on Chef. So foe example configuring a virtual host in apache server. The list will indicate first a package to be present then apache configuration files to be written and finally turning on the service. This helps in code readability and understandability. An example recipe of task described above is shown below.

```
package apache2 do
action :install
end
web_app "my_site" do
  server_name node['hostname']
  server_aliases [node['fqdn'], "my-site.example.com"]
  docroot "/srv/www/my_site"
```

```
end
service apache2 do
action [:start, :enable]
end
```

- **Attributes**: Attributes are simply data values that describes the properties of Node. Attributes used for providing different values to different machines depending on location, purpose or other metadata. We can set Attributes in cookbooks, nodes, roles and environment. This can be thought of as an abstraction mechanism that separates the data from the implementation. In essence Attributes are a special key-value store called a Mash within the Ruby DSL context. A Mash is just a Hash where the key can be either a Ruby symbol (:key) or a string ("key"). The keys are attribute names and those can have some default set but they are always modifiable. All Nodes(host) also have attribute like IP address, hostname, loaded kernel modules, version obtained from ohai but these attributes are extendable by definition of json file to include user-defined attribute as well. On each individual run of chef client it collect attributes values from cookbooks, node, roles and environment and built a complete set. An example of attribute used in apache cook book file located under cookbooks/apache2/attributes/default.rb is given below.

```
default["apache"]["dir"]          = "/etc/apache2"
default["apache"]["listen_ports"] = [ "80","443" ]
```

- **Cook Book**: Cook book collection of attributes, recipes, custom resources, and definitions to configure a certain application or service. Cookbook forms the fundamental forms of distribution in chef encapsulating each and every thing. Hence a cook book written can be shared to other chef users who can make use of it by changing the necessary thing specific to their environment. Cook book contains various components , each of them can be a file or directory. Cook book can be create by i and also can be downloaded if there is existing one for our use case issuing a knife command. Following the directories and files under a single cook book directory.
  - attributes/for storing the values to be used on node basis
  - definitions/ to create and storing the reusable collection of one or more resources
  - files/ build in resources definitions
  - libraries/ To store the helpers and extend chef through ruby code
  - metadata.rb recipes, including dependencies, version constraints, supported platforms and more.
  - providers/ To create and store custom resource provide
  - README.rdoc

  - recipes/ Resources to be manage, in the order they should be managed.
  - resources/ Custom resource declaration
  - templates/ for creating configuration files that are dynamically created by replacing variables with our values.

- **Node**: Nodes are the client machines managed by Chef. Nodes constitutes two things runlist and sttributes. Runlist contains a list of recipes and roles. Normally only a node represents a single machine where these recipes and roles are applied. Node a dynamic property which get build and destroyed. when a client process is started, Ohai detects the information of the host like fully qualified name host name etc.chef client now compare these data with the last known state data of the node from chef-server. Now all the ohai attributes get update to the recent values and extra attributes added via json also get updated. Lastly all the attribute in cook books also get accumulated. Thus the process if node building takes place. With the help of these attributes , runlist and Roles which is expanded into attributes and runlist at run time ,chef is able to apply the recipes on the node that brings it to the desired state.
- **Role**: Roles provides means of grouping the nodes with similar functionality. Role are used to express the parts of the configuration that are shared by a group of Nodes. Roles contains set of attributes and run list. One or more roles can be applied to a node signifying to implement the recipes in those roles. Hence If a attribute and runlist is applicable to more than one machines in infrastructure , we can create a role and assign that role to those machine that needs those run list to be executed. Roles can be created by utilizing REST API , ruby DSL , Knife or through web UI or directly creating json in chef repository. A simple way of role creation using ruby declaration is shown below

```
name "webserver"
description "Simple Web App"
run_list(
  "recipe[apache2]"
)
```

- **Ohai**: Ohai collects information about the platform on which it is running. It can be used standalone, but it's primary purpose is to provide node data to Chef. Information like hostname, FQDN, networking, memory, CPU, platform, and kernel data is obtained by ohai. When ohai is used with chef the information it collected is reported back via "automatic" node attributes to update the node object on the chef-server. Data collected by it can be compared to puppet facts and CFEngine classes.

Working Mechanism

Chef -servers stores all it's configuration files like cookbooks, receipe, attributes files. Normally these files are checkout from version control system to the the directory /etc/chef. The process of convergence starts from clients. The chef client process in the nodes runs every 30 minutes. Hence when it wakes up it will gather the properties of platform using ohai. The ohai then transfers the OS specific information as node attributes to the chef-client process. Chef client now fetches the previous information about the host from Chef server , any additionally json attributes if declared is added together with the generated ohai attribute. Now node building phase is completed hence at this point chef client is able to tell the properties of node.

Now chef client performs the synchronization of all the cookbooks and related files from the chef server. For synchronization with server it needs to be trusted by the server hence the process of registration and authentication takes place with the help of key exchange. Chef-validator is a special purpose client used exclusively for registering new clients. The private key generated after successful registration is kept in /etc/chef/client.pem. This key will be utilized for future authentication. After synchronization gets completed the client will have all the things it needs to execute in order to get the machine to desired state. Thus client process now assemble the specific collection of resources following a definitive pattern of first loading the libraries followed by loading attributes and then after definitions and finally recipes. Now the execution process takes place by applying on node attributes values on decision making statements in the cook books and recipe. Ideally with one execution cycle the chef should get the machine in desired state thus convergence is achieved. After converging, Chef saves the state of the node to persist it's node data and make it available for search by reporting it back to Chef Server.

2.4 Community

Community and documentation support of product is very much important factor for the success of the open source project. All open source projects are made alive by the user contributing to the development and users implementing them and providing support. Community support can be divided into two half i.e. one for the development of the product while other to supporting the deployment and usage of the product. With out taking these factor into account the analysis wouldn't be complete and usable. Often open source project treats it's user as a partial developer. Normally these users implements the product in real environments and reports the bugs exposed and bugs encountered. Some bugs are hard to find in the testing environment of the company , those are hidden bugs and these bugs can show up while the product is implemented in the different environments. Hence open source projects carries out majority of testing by realizing a beta product to the community and organize it's development on feedback and bug reports obtained. The users not only reports the bugs but can supply fixes for bugs they encountered hence involving

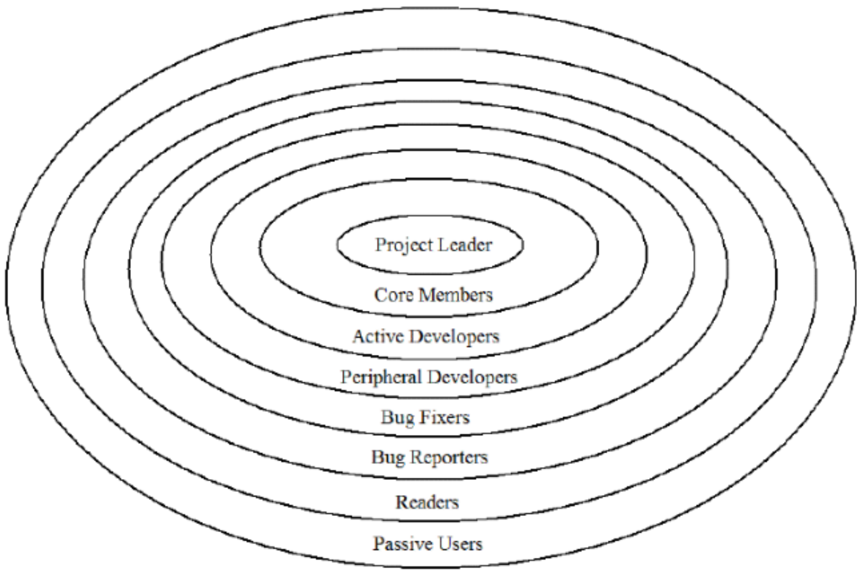


Figure 2.4: The onion model (Ye and kishinda 2003 [19].

partly in the development activities. Also these users using the product creates a helpful social environment through different communication channels like IRC or mailing list to discuss their problems and helping each other to solve problems. The community grows in size as the products get used by the new users and old users provides helpful tips to the query of new comers. New users often stumble up in the help and support forums to get their problem solved. If and when they feel like the suggestion they obtained were helpful they tend to keep up with the on-line community of the product. Normally the community model of most of the project can be represented by "onion model" described in open source literature[19] where members fall in different layers.

This model clearly shows that there are large number of passive users i.e. observers that use the product only get less involved in the development. As the user gets more experienced with the product and the person's knowledge about the product grows they tend to shift more into inner layer of onion model. Also the power of open source project is the people can transition between these layers freely as per their interest and necessity. The onion model however largely represents the developer community. If we focus only on the passive users area only, then we can find the similar kind of layered behavior between the users using the product. While some users being the power user raises a lot of technical queries about the product and creates a discussion around it , other users are only the normal users hoping to get some answers for the problem they encountered hence a small discussion can be created around it. So as the users become more experience with the products and starts to use it largely then the normal users will transition into power users. We believe that the online support of these products also rely in these very power users for answering the queries of the other users. The study conducted by Karim R. Lakhani and Eric von Hippel on Apache user group also shows the similar trend [20]. So it is desirable to figure out if there is some similarity between the user groups with in these configuration tools , have close view of power users and normal users and analyze their activity.

2.5 Reliability

Reliability measures how often a a software works as expected and produces outcome. Hence reliability can be a hard thing to measure as it depends on how a software is used, under which environment and on what the work load. However when reliability id defined as

probability of failure free operation of a computer program in a specified environment for a specified period of time

This definition [21] portray reliability as a measurable metric. Various reliability growth models have been used since late 1930's [22] to measure and predict Mean Time To Failure (MTTF), future product reliability, testing period, and planning for product release time.

2.5.1 Models for Reliability

The models used in software reliability growth model are concerned with measuring the time between the failures and fault count in a known time span. Mean Time Between Failures model expects a set of data that represents the time ranges of successful operation. A distribution model is applied in obtained data set so that a pattern of time ranges due failures and they get fixed can be observed. Finally parameters for the applied probability distribution is obtained through simulation and testing. As the numbers of failures decreases, the reliability is increased. Jelinski-Moranda[23] and Littlewood models [24] are two such examples of this kind of reliability modeling.

In fault count model, a number of faults in a specified time period is used as data set. A probability distribution function can be applied on the the failure rate which is defined as number of failures in a given time. As the fault count decreases ,when the bugs are found and fixed finally increasing the reliability. The fault count show as certain pattern for which a parameter values can be calculated for applied theorotical distribution model. Goel-Okumoto [25] and Musa Okumoto are examples of this model.

Additionally there are two model for gathering the data about failures namely white box and black box models. White box model is used when every thing about the software is known i.e. it's process flow, it's components and structure are known. Thus software reliability is focused on measuring the failure rates of components and relation between them. Black box model assumes the whole software as a single entity ignoring it's various components and interdependencies. The black box model needs the failure data collected over time. Therefore in this case the black box model is more suitable to analyze the reliability.

Generally failure rate can be represented by bathtub model [26] [27], this model is capable of representing the failure rate for anything for it's life time i.e. all three phases consecutively , decreasing , constant and increasing. When a product is new some of it's aspect will fail but the product will adjust itself to the environment and finally when the product is old, the failure rate is high again. The Weibull distribution function is capable of representing all these three phases of bath tub model and also been shown by number of studies that the Weibull distribution function can model the reliability pattern of most of the products [28] [16].

2.5.2 Theory

A failure of software is the inability to produce desired results, this is caused due to error in software, thus an error is a phenomena used to describe when the output deviates from the expected out put. An Error in software is introduced due to existence of faulty component or logic with in a software. Thus this faulty part or defect is termed as bug. Failure behavior of software can be analyzed by utilizing the both PDF (probability density function) and CDF cumulative Distribution function [16]. PDF *f(t)* shows the concentration of data samples in the measurement scale so that the area under the curve is unity while CDF t*F(t)* shows probability of random variable T, i.e. a probability that the value of *T* will be less that or equal to the specified value *t*.

$$F(t) = P(T \leq t) = \int_{-\infty}^{t} f(x)dx \Rightarrow f(t) = F(t) \tag{2.1}$$

Thus *f(t)* is the rate of change of *F(t)*. Hence if T is regarded as a failure time then , F(t) can be regarded as the probability that system will fail at time t i.e. unreliability at time t. Consequently Reliability *R(t)* can be defined as a probability that system will not fail by time t, i.e

$$R(t) = P(T > t) = \int_{t}^{\infty} f(x)dx \Rightarrow R(t) = 1 - F(t) \tag{2.2}$$

Additionally a large number of empirical studies on software projects follows a cyclic pattern described by Rayleigh distribution function represented by a special kind of Weibull distribution function with shape parameter $\beta = 2$. A Weibull PDF represented by

$$f(t) = \frac{\beta t^{\beta-1}}{\alpha^{\beta}} e^{-(t/\alpha)^{\beta}} \tag{2.3}$$

with different shape parameter β and scale parameter $\alpha = 1$ is shown in the figure 2.5. The scale parameter α is responsible for squeezing or stretching the distribution. The distribution function takes different shape on different values of β, when $\beta \leq 1$ it represents and exponential distribution , the distribution becomes bell shaped at $\beta > 1$ and represents Rayleigh distribution at $\beta = 2$ and as β increases , steeper the curve will be.

As seen Weibull PDF with $\beta = 2$ and greater can represent the bug life cycle of the product as when the development of software is considered finished, it enters in testing phase where a large number of bugs might be found

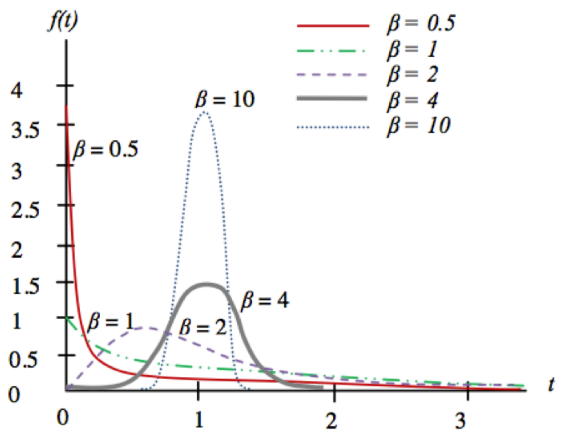


Figure 2.5: Weibull PDF for several shape values when α =1

and fixed. But still there may be more bugs which are difficult to be detected in the lab testing environment and hence these are caught by different users in their production environment due the wide range of use case. Therefore the bug numbers increases slowly when product development is finished and reaches some maximum, then bugs numbers decreases exponentially as test and usage continues and they get fixed. Finally it stabilizes at one point of time but still, there may be certain bugs that might show up which will decrease the reliability. Hence every software follows this patter of bug life cycle , the only difference is how long the bug continues to show up down the time line.

2.6 Usability

2.6.1 Usability as Quality

Quality is a broad term which has many different views of what it means for different people. But often quality is generally treated as a property of a product. ISO 9126 takes this approach and categorizes the attributes of software quality as: functionality, efficiency, usability, reliability, maintainability and portability [29]. ISO 9126 stands as guide for software product evaluation that resembles most of the aspects from the assessment methodologies described in section 2.1. ISO 8402 lays out quality in it most clear and concise form , it states "Quality: the totality of characteristics of an entity that bear on it's ability to satisfy stated and implied needs.". It assumes a set of requirement of user fulfilled by the product, however if there are different requirement of the users then the they may require different characteristics for a product to have desired quality, so that assessment of quality becomes dependent on the perception of the user. And this is exactly what is perceived as the quality in real use case.

Mostly user perceived quality is regarded as judgmental and inaccurate hence not included in the software quality assessment process. A set of qualitative methods and a good software engineering practice followed while development of the product helps to achieve all the attributes that helps to make software qualitative but cannot ensure the it's quality in the hands of the users. Thus there are more than one reason why the user perceived quality is needed to be accounted. In our case , the functionality, usability and efficiency attributes required by a system administrator in a data-center may be very different from those required by an system administrator of a university infrastructure. Hence "Quality of use" [29] is one of the mechanism that allows to measure effectiveness , efficiency and satisfaction of users with specified tasks in specified environment. "Quality of use" refers to the extent to which a product satisfies stated and implied needs when used under stated conditions. Any relevant aspect of software like functionality, maintainability , reliability etc may contribute to quality of use. Hence the measures of "quality of use" can be used as criteria to determine whether or not the product is successful in achieving usability. Thus Quality of use i.e quality of user experience thus provides a means of measuring the usability of a product which up until now seemed to be a difficult and almost impossible metric to be measured for a software.

2.6.2 User Experience Measurement

If we agree that a software considered is capable to do the designated task then it's usability now is extremely depended on users experience , there is no single universal method to measure this user experience. If asked the to users of product on how they solve their problem utilizing the software , it can be largely subjective and biased. Thus to measure usability in a scientific way a series of metrics can be collected to do the same task that can give help us measure the quality of user experience, finally allowing us to characterize some thing has good or bad usability.

User experience can give us the usability as a measured unit including both subjective and objective aspect. The objective part should take care of all the factual metrics like task time, stress , efficiency etc. associated while user is performing the task , while subjective part accounts the user feeling towards execution of task using the software. But objective data and subjective data collected from user can point in the same direction i.e. have positive correlation which is a good indication of successful usability measurement, while negative correlation reveals disaster either partly due to user not interested or distracted on subjective data collection phase.

An example of negative correlation is a user spending large amount of time to carry out a task using a certain product , rating as a favorite one for him. In this case the user's expression contradicts with the deeds and does not give a meaningful result. Hence if a test is carried out with a set of users to measure user experience , both objective and subjective need to be collected and analyzed separately which will give meaningful insights into problems , productive time, stress level etc using the product. Further calculation like correlation of the data sets to can be done to measure the effectiveness of test. All these things combined together will give a good measure if user experience.

Combining objective and subjective method helps us to compare and analyze following things

- Two or more similar products used by same set of user to carry out same task under similar environment
- Two or more users using the same products performing same task under similar environment
- Two or more task performed by same set of users using same products under similar environment

2.6.3 Objective Method

The objective method help to report factual information about the task performed by capturing various data associated with task. It helps to measure the effectiveness , productivity and efficiency of product for user. Various type of metrics are collected by the usability testers depending on the type and interaction mechanism offered by software under test. For example majority of the usability test is carried out for GUI application , hence testers gathers the key stokes , numbers of clicks , number of back presses etc. But thw common metrics used in all types of usability test are completion rate, task time, problems encountered and errors made etc, these metrics can be used in any kind of usability testing regards less of product type. Following are the metrics that can be collected in order to carry out objective method of usability measurement.

- **Completion Rates**: Completion rate are fundamental metrics collected that provides a simple metric for success. These metric is collected in binary form i.e. completed (1) and incomplete (0). If a task is in complete , no more measurement needed to be done, apart from collecting the reasons for task incompletion. Task completion is widely used in usability testing as it is easy to understand for readers and report. Prior to a test a a set of criteria needed to be defined in order to consider a task as complete. In task completion focus is given on the output obtained by user which is later compared with success criteria , rather than the path taken by user to complete the task. If the result does not meet the criteria , no matter how close was the user near to task completion it will not be considered for any further evaluation. The average task completion rate is 78% [30]. Hence 78% task completion can be used as reference point to analyze the results. Any thing above 78% can be regarded as above average and a nice task completion rate. Since this completion rate measured was the result from the commercially available software's on thousands of users, it has raised the average completion rate scale. Another cause to have such a high rate of task completion is due to the mentality of users in test , they know that they are being observed and show a high tendency towards task completion.
- **Task time**: Keeping track of the task time helps to measure the efficiency and productivity. As completion rate , task time are also core metric to

be measured for usability study and widely used in majority of usability test. Normally the task times are measured in seconds and minutes by keeping track of start time and completion time. Task time can be measured in three ways [30]

- Average task time: Only considered for users that has completed the task.
- Mean Time to Failure: The average time users took on task before giving up or completed the task
- Average Time on Task: The total time users are spending on task.

Task times are also helpful for diagnosing the usability problem. If a user is taking long time to finish the time it can often be due to the problem faced by user or errors associated with the product. When average task time, is considered this studyshows that for a small sample size 25 or less , the geometric mean will be more accurate measure than arithmetic mean or median. Also since task time data set are positively skewed as some users take more time to complete the task, the arithmetic mean will can be dragged to higher values and this also is the main reason for task data distribution not being a normal distribution. By separation of time the user spend in doing the task from the total task time, we can thus obtain the productive time. Unproductive period of the task are the times when users spend their time searching for the help and understanding the structure of software , or overcoming the problem faced. Hence Productive period can be calculated by removal of these unproductive period from the task time. These productive time can be crucial for product comparison for performing same task.

- **Usability problems**: Keeping track of problems faced by the users helps to calculate the probability of users likely to face the problem, More over it can help us find unknown problems associated with the products. If the record is maintained such that it can classify which user faced which problem , it helps to measure the problem discovery rate and most importantly predict a better sample the sample size that can give us reliable result [30]. For example

  We need to test on average 5.32 users to discover 85% of usability problem given the occurrence of problem is 30%.

  The theory behind this claim is used of binomial probability distribution as stated in [**?**] and derived in [31]. The author of [31] has achieve the result by performing the calculation shown in. The binomial probability function is applied in detection of usability problem such that n represents the total number of users, r equals number of problems and p equals the probability of occurrence.

$$\frac{n!}{r!(n-r)!}p^r(1-p)^{n-r} \tag{2.4}$$

For no problem to occur at all,i.e. obtaining p(0) we can replace r with 0.

$$\frac{n!}{n!}1(1-p)^n \Rightarrow p(0) = (1-p)^n \tag{2.5}$$

Therefore we get the probability of finding at least one problem by subtracting the probability of finding zero problem from one.

$$p(\geq 1) = 1 - (1-p)^n \tag{2.6}$$

Additionally the author [**?**] also has shown that 5 users is enough to show the majority of problems through the used of binomials.

- **Conversion**: Conversion is a special kind of completion rate. This is also a binary measures i.e. (0= not converted, 1=converted). By conversion , we can think of user carrying out a series of task to get the ultimate goal. For example in our case if a users can download a product and implement it to do automate or configure a part of infrastructure then we can say the user have converted. Thus it covers all the phases of usage of product and will give important information about on users being actually able to solve their need with the product..

2.6.4 Subjective Method

Subject method helps to capture the user perception on the product by asking a user to answer a set of questionnaire. If a user is happy about a product after using it , he/she is sure to give positive comments for the products on the other hand if the product is frustrating to use , with user facing too many problem in every turn while using it, it will definitely raise the negative feeling for the product. Subjective measure helps to capture those feelings and present result in measurable form. Thus it can also be said that it measures users satisfaction level. A satisfaction is composed of comfort and acceptability of use [29]. Comfort refers to the overall emotional response to use the system (whether the user feels good, warm and pleased, or tensed and uncomfortable). Acceptability of use refers to the overall attitude towards the system which covers whether users feels the system was able to solve their need, if they feel in control of the system while using it etc. This satisfaction can be measured in two levels while conducting the test which are discussed below.

- **Task level satisfaction**: It helps to measure the difficulty level of the task. Popular method of evaluation of task level satisfaction are ASQ(After Scenario Questionnaire),NASA TLX(NASA's task load index is a measure of mental effort), SMEQ(Subjective Mental Effort Questionnaire),UME(Usability Magnitude Estimation) and SEQ(Single ease question)[**?**]. Getting users to answer a questionnaire immediately after performing the task is simple and reliable way of achieving task-performance satisfaction level. Thus task level satisfaction can also be termed as performance satisfaction.

- **Test level satisfaction**: It measures the user perception towards the product. Since it is asked at the end of test and the questionnaire is focused on getting the general information like how was the product , do i think i can learn it quickly etc , it helps to capture the overall attitude of users about the product. Thus it is also regarded as measure of perception satisfaction. System usability scale (SUS) is one of widely used method to compare the usability of product. There are other measuring scale as well as home grown proprietary method for carrying out this task, but the SUS provides high level subjective view as questions are quite general. More information about the SUS and it's structure is provided below. The task level satisfaction definitely has impact in the out come of it's result but it's outcome is not solely driven by task level satisfaction and slight difference exist between two, how ever there is also a strong co-relation ($r \geq 0.6$) between these post task ratings and post test ratings.

Single Ease Question [SEQ]

As discussed in 2.6.4, asking user about the task they just carried out at the end of task provides useful information. But a care must be given such that post task queries should not be long and complex. It can effect the users answering mentality if it unnecessarily long and difficult to answer. At same time post task question should also be capable of capturing the user perception towards task such that a task performance can be measured. Therefore a good questionnaire should be

1. Short
2. Easy to respond to
3. Easy to administer
4. Easy to score

SEQ demonstrates all the above traits [32]. It contains only one question "Overall, this task was ?" and the response is the scale that ranges from 1 (very difficult) to 7(Very Easy). The strong point of SEQ is that it is observed that user builds their expectation into their response. So when the task steps is added or removed , users are seen to adjust their expectation on scale accordingly. Also the out come of SEQ seems to have high correlation with the outcome of the post test questionnaire i.e. SUS [32].

System Usability Scale [SUS]

System Usability Scale is a ten item questionnaire that provides a high level subjective assessments of the usability study. It was developed by John Brooke [33] in 1986 as a tool to be used in usability engineering of electronic office systems. SUS is technology independent and since it was developed has been tested on wide range of platforms ranging from hardware software , websites, cell phone etc. Thus it has now become an industry standard. Each question in SUS has a 5 response options. This 5 options acts as a range scale , giving user the flexibility to respond accordingly. The 10 question are as follows and the answer options is a range scale from 1(Strongly Disagree) to 5(strongly Agree) for these question.

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

SUS question are highly general and capable of giving a measure of perceived satisfaction. But a research [34] has also shown that SUS is capable to give a measure of usability and learnability revealing two dimensional aspect of SUS and these two factors actually do correlate [35]. Particularly the questions at 4 and 10 point to the learnability direction, therefore the SUS score for these questions can be compared among product to compare the perceived learn ability by users. The SUS scored then needed to be calculated for the response collected in order to make a sensible interpretation hence following process need to be carried out for calculation of SUS scores.

- For odd question numbers, subtract one from user response
- For even question numbers, subtract user response form 5
- The values are now in 0-4 scale range , where 4 being the positive response
- Lastly the total response of individual user is calculated by adding the responses and multiplying the result with 2.5. This converts the range of possible values from 0-100 instead of 0-40.

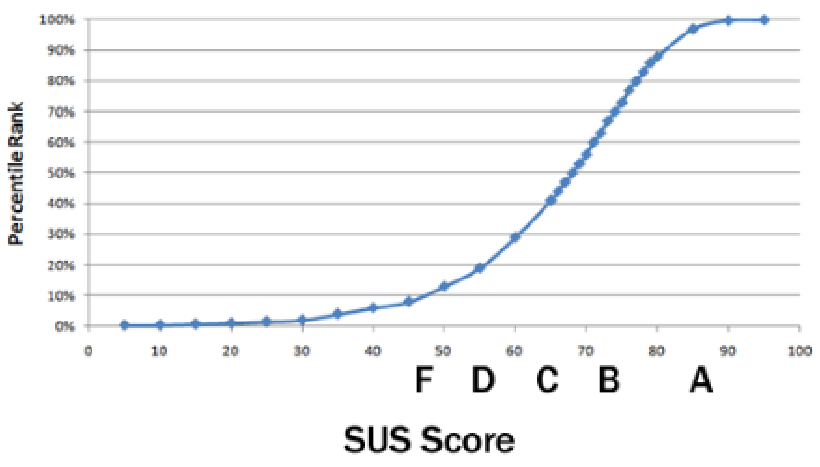


Figure 2.6: SUS score percentile association and corresponding letter grades

A SUS score range from 0-100 is not percentage. The best way to interpret a SUS score is to convert the individual score into percentile rank. The figure[30] below shows the percentile ranks associated with SUS scores and the grade letter.

Now with the help of figure 2.6 it becomes easier to interpret the score obtained from the SUS score. The average SUS score from over 500 studies is 68 [30] i.e. 46% which gets a letter grade C. Therefore product with grade C are considered average , anything above is considered as above average. For example if a SUS score 74 have percentile rank of 70% , there fore this product have a good perceived usability than 70% of the products tested by the SUS. A SUS score of 85 or more helps a product to get a letter A , these are the kinds of products that are likely recommended by people [30].

Chapter 3

Approach

3.1 Community

Software aspect like community always have a vague definition but is increasingly important for support and success. Though it is an important criteria that needs to be analyzed, often the previous studies conducted has failed to grasp it into account and give a reasonable overview of the community activity of the three products. Often there are comparison on these tools that states about community [36], [37] but does a very little research on it. Measuring people in IRC is not only the way to measure the community strength neither the numbers of download of the software packages from the repository of popular os platforms like Debian ,Ubuntu etc. Rather it can be a part of measuring the over all strength. Community of software is also deeply associated with popularity of the products, but all the products seems to claim itself as a popular and dominant player in market. Again if a product is popular and have large community, it must have large number if resources available and discussion around. No prior research had attempted to identify or project the resources available and make comparison based on it. Mechanisms like getting the metrics from meta forge, help sites like stack over flows, the mailing list and determining how soon a question get answered. Articles as well as documentation provided can be used to measure the community strength.

The community of the product can also be divided into two main groups , i.e. the developers group while other are the users of the software. Often the discussion of the developers group are focused in the product development and bug fixes , while the users group are focused in the implementation of the product addressing different problems. Together these group help the product to grow , become stable and better over time. We will be concentrating on the analyzing the users only group activity and try to figure out the puzzles related with community size and behavior of the members with in. Inspired by work done in [38] [13], a reasonable view of the community size will established. Similarly popularity of each product will be studied carrying out some activities discussed in [39], [13]. By understanding the community behavior from the data gathered, a mathematical model will be utilized to find out if there is any difference in community despite of their sizes between these products. We will also have a look at the discussed issues and have a relative projection of what problems a regular user are facing while utilizing these products. Having a close look at the community and popularity helps to understand the field support we might get for the product.

To state it in brief the community study in this paper is focused in the

- establishing the popularity of individual products
- the resources and materials available for each products
- analyze the community and it's behavior taking a mailing list as primary source of information for these projects
- having a brief look at the topics and problems the community are talking about

.

This study is hoped to be helpful for possible new comer to understand the community and have a view of topics and problems discussed in these products. It is also helpful to the respective project to view it's community behavior and make improvements on it. Also by focusing on the problem areas that people are regularly facing ,the product qualities can be improved and made more user friendlier. The paper investigates the community starting with analyzing the popularity ,gradually moving towards analyzing the online support for these products and finally establish a relation of the social support behavior of these products with a mathematical model if suitable.

3.1.1 Popularity analysis

Top view

The primary source of investigation of popularity will be google and different social discussion like Hacker news. We care for these source because people look for these sites to get the answers for any queries apart from posting their queries on mailing list. Additionally people also share their experience with these product and have a discussion around which is very useful to a new comer to get a good understanding of merits and demerits of product. Google being the most popular search engine, it keeps the records of keyword searched. We can get an idea of popularity by analyzing the trend of frequency of search terms related to these products in Google. But it should also be kept in consideration that the result returned by google isn't picture perfect, there might be noise in data as these key words could be associated with other searches and not at all related to configuration tools. So necessary filtering mechanism needs to be applied, so that it can be assured that majority of result returned by the google is related to configuration tool and the keywords actually belongs to some topic related to configuration management. The various tools from google like google insights , google correlate will be utilized to get a over view of trend analysis. It is also interesting to have a overview of web talking about these product so by looking in the numbers of other websites referring the main web site of these products , and social discussion and development forums an estimation of community ecosystem of these products can be made.

Focusing on the area of interest

Now a days a lot of social news websites that enables news aggregation are available. People come here often to talk about the the things they see else where. While the mailing list tends to be useful for finding the answers problem related to the particular problem, these kind of social news provides the platform where user talk about their experience with the product and any other thing related to that product. For example if there is article posted for the product , individual can discuss around the usefulness and validity of article as well as it also addresses that article. Also blog writers usually submit the articles they posted on their blog to sites like Hacker news to get a much wider range of audience. Thus it is more sensible to analyze the social news site to make most out from the discussion happening in the web for these products. Having a view at discussion activity level on these forums can give an rough idea of peoples using these products and the resources that could be found for the product. If the data in these forums is analyzed one can possibly find the advantages and limitation of products. But for now the focus is on calculating the activity level of the community regarding these products. For this purpose popular social news named Hacker news was analyzed.

Estimation of population

As part of analysis of data from social news sites like hacker news, a set of samples will be collected at random. Each samples will collected a from set of result will be analyzed to give us a count of numbers of false positives observed i.e. the data that doesn't belong to system administration and IT as a whole. This presents us with total number of results found with total number of data analyzed. So a sample proportion can be calculated and utilizing this sample proportion and estimation of total population proportion can be made. Finally a a value will be estimated from the calculation and make hypothetical test for testing if estimated value represented the population proportion. If it is satisfied we will be using that proportion of population as the result we obtained. For this method to be utilized the sample should contain more than 10 correct results and more than 10 incorrect results and the samples must be drawn at random. Also the population size must be 10 times the sample size. We will be fulfilling all the criteria when we gather and analyze the data. All these task will help us understand popularity and resource availability of these products.

3.1.2 Community and Support analysis

A social environment involving a product is created when people using the product help each other. Apart from solving each other problems they might discuss set of features they wanted a product to have. Either way mailing lists are the primary place where all of these kind of discussion happen. Mailing list of the open source project are kept alive by the user for the user. Thus by analyzing it several things can be understood about the software. In order to have a clear understanding of support that can be obtained from the community , a close analysis of mailing list will be done. The things like how many users are subscribed and their activity can clearly be perceived by glance but a in depth look into the interaction happening in the mailing can help us understand the community structure , the quality of field support provided by the community , problems that are frequently faced by users, topics the community are talking about etc. So the focus will be to have in depth look at the discussion happening in the mailing list to understand the community structure of each product and the support they provide. Since the study is also concerned with analyzing the community support ,the mailing list makes itself as ideal source that is primarily used for discussing and placing problems for using the product.

As pointed out by the results from past studies and research conducted for understanding the community , a community of users can be divided into different groups depending in their activity level. In a community of users, different users have different activity level. Some users tend to be passive describe by users subscribed to the mailing list and only posting problems while others are active i.e. subscribed in the mailing list and keep a close eye on the discussion happening and frequently replying to the queries and taking part in discussion. Inspired by the work [20] we will try to categorize the users into three groups and study their behavior. The task of grouping the users have several advantages , it helps to understand what portion question is asked by what kind of users, what portion of question are being answered and is there a difference in answering behavior for the question posted from different groups. Additionally answers to the question like which groups of users are mostly giving answers and how big is the group can be obtained as well. A part from that it also provides a clear view of total subscribes divided into groups on basis of activity they perform.

Community structure

The approach to classify the users is purely based on the type of activity they perform in the mailing list. Basically there will be those types of users in mailing list one , that places the questions only i.e. opens a thread. Another type of users are both involved in placing the question and answering the question , while other type of users are involved in answering the question and taking part in discussions only. So users can be categorized as follows

- **Seekers**: These are the novice and passive users. They are involved only in posting question in the mailing list and look up for mailing list for their question to get answered. These users are the future prominent member of community i.e. as they grow experience they can take part in answering others queries.

- **Seekers and Providers**: These are the users that places the question and equally replies to other users questions. These are the kind of users that make the community support alive. These users are experienced and are always sharing their knowledge with others users. They are also equally benefiting the product by posting queries that helps the product to grow. A part of these group that has a heavy traffic can be termed as power users.

- **Providers**: These are the kind of users that provides solutions only to the queries posted in the mailing list. These users are can both be passive or active because person can be subscribe to mailing list and is occasionally answering the problem that he/she has faced utilizing the product while their can be other users that are frequently answering the queries i.e. the product creators etc.

Supportive Activity

By dividing the users into above discussed groups a number of things can be understood about the community of these product. A set of metric was focused for which a data was to be gathered for all the groups. However we will be focusing in analyzing the activity of seekers only and seekers and providers. These metrics were chosen to get a meaningful result out from the mailing list data. The set of metric for which the data was to be gathered is listed as follows and all of them are to be collected as percentage to represent the proportion of total data.

- Subscribers: It represents the proportion of users falling in the group while at the same time time giving a visualization how big group making it easier to analyze the activities performed by this group.
- Questions posted: It represents the portion of query posted by the group. It helps us figure out how much of the total queries in mailing list arrives from this group.
- Answers supplied: It represents the portion of answers supplied by the group. It helps to know the importance of groups's role in providing the answer to community.
- Unanswered question: It represents the portion of queries asked by the group that goes unanswered. It helps to understand the community behavior for answering the queries from this group and how much could you expect from the mailing list if one falls in this category.
- Answers in 1st day: It represents the portion of question posted by user from this group that get answered in the 1st day. It measure the responsiveness of the community , additionally helps to see the responsiveness compared to different group.

Benefit of representation of metrics as percentage of total data are it is possible to take the whole data in mailing list into account at the same time it will enable us to take uniform metric measurement for all products despite of the amount of data in their mailing list. For example if there are large number of users for puppet , the numbers of threads will be high as well, so it makes no sense with measuring the numbers and comparing with other product like chef or CFEngine. Additionally all of these products were introduced in different years hence some might contain a long history while short for others. So just looking the numbers can miss lead the analysis that will be done on the statistics collected.

Topic and problems discussed

Finally the subjects of each threads will be analyzed to get an understanding of topics discussed and the problems frequently face by the users. Since a particular problem ideally is represented by single thread , if different threads with describing with same problem exist in the mailing list then we can understand that the problem is frequently faced by the users. Thus analyzing the threads can give a good idea of the topics that are frequently discussed and problems raised. For performing this task , the subject of each thread will be analyzed by implementing n-gram counting program to extract text from these subjects. Thus combination of words with high frequencies can be regarded as topics/ problems frequently faced by the problem. The differentiation of weather the word combination refers a topic discussed or is associated with problem is done through analysis of words and it's combination with other word.

3.2 Reliability

It is always desirable to know how often a products fails but difficult to measure. In case of configuration management failure becomes key concerns as system administrators rely on these to to carry out automation and consistency with in the system. This means if failure happen in configuration tool then it is more likely to break the whole system , ultimately leading to down time. And this directly contradicts the work the configuration tool is suppose to do. They are responsible to bring the system back into functional and operational state in case of failure happens, and having a too many failure in configuration tool is not a option. They must be resilient as far as they could and should posses least number failure rate. Thus this paper is concerned about how often the products fails and study the failure pattern, and what is it's impact on reliability.

There exists several model to carry out reliability analysis [27], in this case fault count model coupled with black box model which focuses in getting the numbers of faults seen in software for a certain period of time will be utilized. Thus the study ignores software structure. The data about failures can be gathered in two ways, either gather the failures through automated test as shown in [40] to get much faults as possible or the failures previous experienced by customers. Thus reliable source to get the list of faults i.e. bugs for a software product previously experience by user is it's bug tracker. Hence the failure data about all the three softwares will be collected from their online bug repository. After the collection of data , a filtering mechanism will be applied in the data so that all the possible noise are reduced. A possible source of noise in bug tracker is that these products keep the bugs related to each and every thing i.e. documentation , other projects etc in the same bug tracker. Hence the data related to the other projects and products needed to be identified and discarded.

All of these products have different maturity level and lot of (minor and major) version have been launched for them. Therefore a favorable start date needs to be chosen for each so that comparison of the reliability growth of last three major version of each software can be done. Finally the filtered and separated data for each version needs to be organized into bug frequencies for a certain time frame, which will be weekly. So a weekly bug frequencies for each version of product will be gathered which can show the bug pattern for software over time. These bug frequencies will be plotted against the time and further analysis is done through curve fitting and obtaining the reliability growth.

Weibull curve will be fitted for each plot using Maximum likely hood estimation method (MLE) and it's corresponding parameters i.e. shape β and scale α will be calculated. Simplifying the equation 2.3, the Reliability is given by [41]

$$R(t) = e^{-(t/\alpha)^{\beta}} \tag{3.1}$$

Thus utilizing parameters obtained from the curve fitting process the reliability at at time T for each version of product can be calculated. This reliability thus obtained also can be plotted against the time which in this case is weekly , therefore reliability plot of different versions for the time range can be obtained. The reliability of each version is compared with another version of the same product so that an reliability growth can be observed for a each product.

3.3 Usability

People have longed to know which if the configuration tool offers better usability. This has always been an disputed matter as each claiming to be easier to use and learn. Puppet claims about it's DSL being simple and easy to understand. With it's top level language hiding the underlying details of configuration tasks, it hopes that users feel more comfortable and productive. Chef on the other hand claims it offers the best user experience as it offers everything to be done by ruby code. It hopes by executing the configuration on the code flow basis separation of different underlying aspects into aspects of it's frame work will help users to write more flexible and extensible configuration codes. CFEngine also have it's own ideology about the usability enabling users control each and every aspect of system but requires it's users to have deep understanding of system. CFEngine puts simplicity and consistency by offering it's declarative language that operates on promise model. The challenge for the study is to find out which of the product is easier to work with. By easier , we hope to find out which one offers less obstacles , which product is learnable, whose documentation is helpful to get over problems in case it happens and gives user a sense of control etc.

The background study was quite helpful to figure out the processes of measuring the usability in qualitative and much more structured way. Since the mostly used metrics that needed to be measured to make a good usability are known, the part of the challenge was ,executing the process of collection of these different metrics. Additionally we need to decide what will be the sample size of data collected , i.e. are we targeting for the huge data collection by conducting the test on hundreds of user or the a data is collected only from the small set of users. The decision of sample size has a huge implication of mathematical models and distribution applied latter on in the analysis of these data. For example, if a average time taken to do a particular task is to be calculated for a set of users for a product , then if it's small sample set , a geometric mean will give much accurate average value compared to median.

Apart from the decision of sample size we need to decide how the usability test is going to be conducted on a set of users , i.e. through a controlled lab set up or through distribution of test so that user can conduct on their free time and supply back the results. The next challenge was to over come was how the data will be collected from users and what will be the methods and process in place to collect these data. These were the problems associate with conduction of test to get metrics from users i.e. problem related to test implementation and data collection.

Another major challenge was deciding the task that users were going to perform. The selection of task was very crucial since it needed to be represent the task that system administrator commonly use the products for and it also need to be the most basic form of task so that it doesn't consume much time(days) of user while performing task. These two aspects of the required attribute of task were very contradicting as system administrators usually performs one huge complex task involving a series of task as per the requirement of their environment spending days in each task. Hence it was decided to conduct the task that represent some sub tasks commonly conducted by system administrators. Thus all these subtasks combined together can represent a bigger task that justifies the use of configuration tool.

3.3.1 Sample size and Test Conduction

A reliable and qualitative results can be obtained even from small sample size. The test conduction required each candidate to be provided with two machines where they can play around with the products and carry out their task. So driven by this use case scenario, for conduction of usability test and achieve a reliable result from small sample, we can conduct test on 5-10 users. The users will be real world system administrators or personnel with a lot of experience with utilizing different system administration tools. On top of that they also have necessary problem solving skills in case if any problem arises when implementing task using a particular product. Thus the sample size of our usability test is around 5-10 and with real world system administrators as usability testers, we can gather a good amount of information with acceptable reliability and accuracy. The data obtained from this sample size can help us analyses the usability on both subjective and objective aspect. As mentioned in background study metrics obtained from this sample gives the some idea and degree of measure of usability but the results should not be overstated too.

For conducting the test, the powerful machine will be deployed to serve the Virtual machines to the users. The host machine need to host a set of virtual machines that we can distribute among users. Proxmox that helps us to create a pool of virtual machines and manage them. It also supports assigning the machines to the users , hence Proxmox was ideal for our use case where we needed to provide a users only the machine they are concerned with. The Virtual machines created in Proxmox will be assigned public ip address so that they are easily accessible from outside , and two machines allocated for each user. The users will be given credentials for Proxmox login , where they could manage their two machines i.e. start stop restore etc. They could securely log into those machines utilizing the public address associated with them. This will offer flexibility to users, machines are reachable and manage able any time. Lastly users will be send an email describing the instruction about the usage of those machines , together with the task list and links to electronic forms from which they can report back their result.

3.3.2 Tasks and Metric Collection

The tasks selected were to represent a each sub task conducted by the system administrator and when every tasks were completed , it give a meaningful result out of it , i.e. some application or service configured using these tool. Since web servers were most common these days and every organization have them, it is common for system administrator to install web servers on machine, ensure it is running and make some company specific configuration so that it is capable to fulfill the company requirement. The most commonly used web server was Apache , hence each user will be assigned the task of installing the Apache making sure it is serving some web site under particular domain name. Over all outcome of the tasks will be a web server capable of serving a multiple web sites. Below are the 3 tasks that will be asked to the users to automated using the three products. Together with the task a list of resources is supplied that can help user complete the task number 1 and 2, but it is up to user to use these resources.

1. Install a server and client of each product, such that client is able to receive configuration from server. (A hello world example will do)

2. Install a software package Apache and make sure it is running in any machine.(In server or in client)

3. Configure a virtual host in Apache, so that it now can serve a web application with domain name nextapp.iu.hio.no.

The tasks chosen addresses different activities exercised in system administration. Task one is aimed at representing the infrastructure set up of configuration management tool. The second task represents the package management and service management. Lastly the third one represents the configuration of installed service to suit the need and thus demonstrate the automation. Together this task combined, it serves a high level goal of having a highly available web server in the organization capable of serving multiple web application.

Additionally the task needed to be performed in order , for achieving the high level , however the users can perform the task in random order just to get each task completed, but the goal cannot be achieved. The listed task also shows the higher level of difficulty as the task number increases. The first task can be easily carried out by user by reading the quick start materials for each product. The second task needs a little bit of searching but can be carried out citing examples. For the last problem the resources are not made available , hence it needs some more effort from user to solve the problem. Additionally since the problem is more specific, it also forces the user to learn about the product in some what detail. Hence if a user can get the problem 3 to be solved for the product then, the novice users had immediately reached to the medium level for that product. Thus it can help us figure out the learning curve needed for product.

Task time and SEQ and Problems rates

The users are send link of electronic form that were made using google docs. These forums have spread sheet as backends. The forums have different fields like start time, complete time etc that user need to fill into , upon submission the data gets inserted in the backend spread sheet. It is a very handy way of data collection from users. Some of the forms fields are required while others are not , so in order for the user to submit the form they need to fill in the values of required fields and others could be left empty . Following are the fields in form that help us to collect task specific information from the user.

- Start time , Complete time for the calculation of task time for the user
- Stopped time for separation of effective time used in task, to calculate un productive period while performing task
- Problems for gatherings issues faced by user and to make relative projection on bugs that user might face later on analysis.
- Errors give the idea of confusing steps involved in carrying out the task
- Single Ease question to get a subjective measure of task and finally calculate task easiness

Apart from these fields it allows users to enter their name , choose the product and choose the task they performed. This form was to be submitted after every task completion. So a single user will fill this form for 9 times. Thus gathered data will be processed into the metrics suitable for analysis. For example , the start time will be subtracted from completion time to get a total amount of time in minutes required to carry out the task. Also if a task has completion time , it is regarded as complete other wise , it is regarded as incomplete. Thus task completion rate can be measured for these products and compared. The scale value obtained for single ease question can be averaged together for users for same task for same product. Thus it can then be compared with similarly obtained data on task easiness for other products.

Problems finding

Similarly the if there are no completion time for users , then what we will have is a list of problems encountered by the users. By this way of data collection , we can now keep track of which user faced which problem thus enables us to calculate problem discovery rate. The problem discovery rate is a negative attribute for product , the higher it is the problematic the product is and it decreases it's overall usability. We already know that 2.6.3 around 85% of problems if the problems tend to affect 31% or more of the users and be found out by carrying usability test in only 5 users. Hence with such a small set we can know the obvious problems associated with products in performing the task , might also catch the rare ones. It also implies that since the sample considered is small , the problems encountered by the users are also encountered by wider set of users in real world because the problems are frequently occurring.

Conversion

When we get information about the each tasked performed by the users, the data can be filtered on users completing the task. A conversion is binary measure and is taken into account when users is able to perform all the 3 task. This also means user is able to achieve high level goal associated with the task. Hence by looking at the task completion for the products , we can separate out the users that completed all the task using the product , thus finally calculate the average conversion rate for the product. In our case if the product with high conversion rate will be extremely easy to learn and understand as well as use.

SUS scores

A link to a form containing 10 question with each question having a rating scale as answer will be send to the users. The users will be instructed to fill in the form and send their replies after completion of all three task for a product. Therefore users will be filling and submitting these form for three times each for a product. The forms are made using google docs , thus SUS scores send by the users will be automatically collected in backed spread sheet associated with the electronic form. The SUS scores thus collected will be process further for analysis. For example the SUS score of individual user for a product will be calculated by following the steps mentioned in background study about SUS on 2.6.4. These SUS scores calculated from users for product will be averaged to get a SUS score for product obtained by this test and finally compared with SUS score of another product obtained in similar manner. One thing to notice is that if the users are experienced with any of the product previously the SUS scores can high towards that product hence record will also be kept for each user about the previous experience with these products.

It is also possible to compare the SUS scores from individual users for different product and study the co relation with their task performance. There need to exist a positive correlation between the SUS scores and Task performance. For example the users can have lot of problems executing the task and still give high SUS scores to those products , this will result a be negative correlation between the task metrics and SUS scores. Therefore by carrying out a co-relation between these data sets we can make sure about the validity and reliability of SUS score obtained.

Chapter 4

Data Collection and Results

4.1 Market share and Resource availability

4.1.1 Usage trends

The figure4.1 shows the result obtained from the google insights. The result shows the search trend for the key words CFEngine, chef and puppet. As seen in the result it shows that CFEngine being the oldest product used to be popular but slowly decreasing the popularity as time passes by while the puppet and Chef are gaining the popularity. At the End of 2011 all of the configuration management tools seems to have same popularity level with some minor fluctuations. Google keeps account of how many times the key words appeared and in what context as well. Given that chef and puppet are common dictionary data , a search on these key word might be coupled with lots of noise. So a filtering mechanism was applied on basis of context that these key words were used in search term. Here we used operating system as context, so that we can be sure that these key words were actually used to do configuration management task on some operating system. The full depth of the context search filter applied to get the trends of the keywords is

```
All Categories ->Computers & Electronics->Software->Operating Systems
```

The fig 4.1 not only shows the time series of graphs involving the frequencies of the of keyword appeared in the search terms but also gives the overview

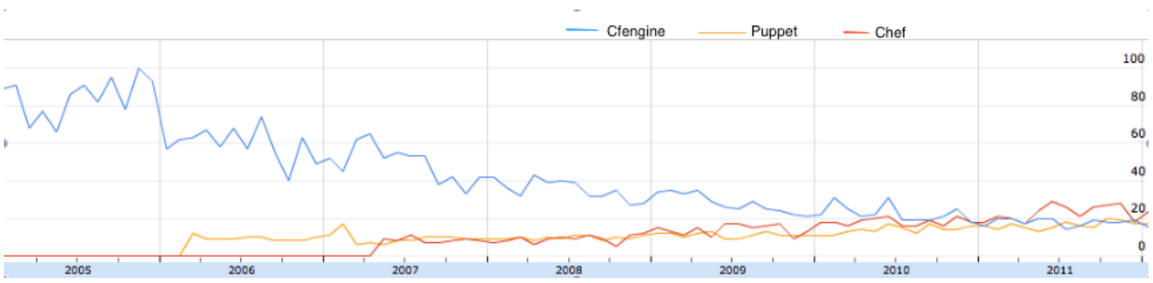


Figure 4.1: Trends for key word CFEngine, Chef and Puppet from google insights

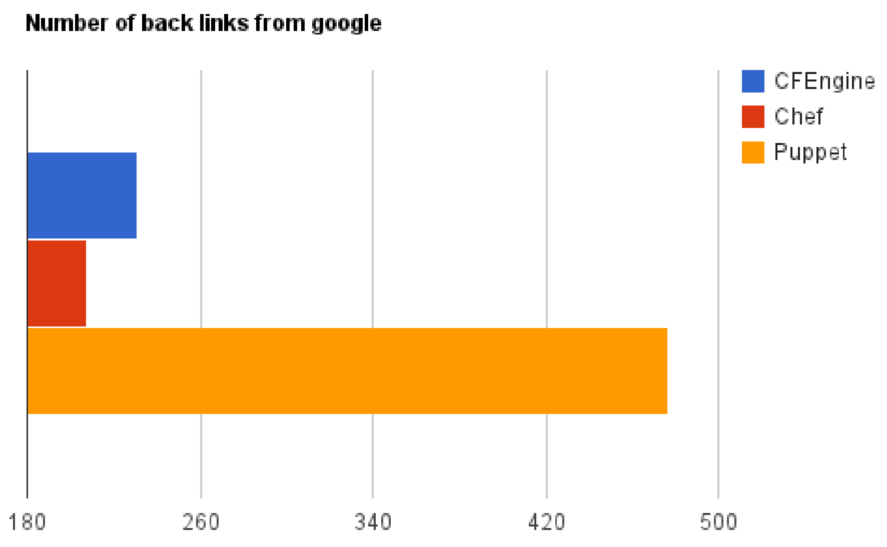


Figure 4.2: Websites referring the products website from wholinks2me.com

area of from where those search were made generated. Here CFEngine seems to be a global player where majority of the searches made from four different country namely United states, Germany , France, United kingdom while puppet and chef seems to be popular with in the united states.

4.1.2 Web site Popularity

Having a view of usage trends we now focus on exploring the available resources for these products. We would like to know how many websites or blogs are out there that talking about these products. These websites and blogs will definitely have discussion on these products and people taking part on discussion these blogs can be referring to the important material that might be useful and contained in the web site of the product itself. Hence regards less of whether it is a marketing forum or a social discussion forum we are interested in achieving the numbers if web sites that are referring to the product web site. In this way we can have a relative guess the of material that we can get from web for these products. Figure 4.2 shows the numbers of back links provide from google and it gives an indication of a large amount resources are out there as compared to that of chef and CFEngine. But we cannot solely rely on this statistics to predict the resource availability or popularity as , it might be a marketing campaign that might has influenced these number.

The number of incoming links is an important part of Googles famous PageRank algorithm. It categorizes the link on various factor measuring it's usefulness and give rank to link of the website. Hence rather that having a look on only the number of web sites pointing to product main page , we have also have look at page rank and figure out the quality of those link backs. It was quite interesting to know that though CFEngine have less numbers of websites referring to them it has page rank of 7 out of 10 while , puppet and chef both have the page rank of 6 out of 10. So it can be referred that even though CFEngine has less number of websites talking about it , the link backs from these web site it quite useful. Hence it can also be conclude that puppet have a lot of buzz created around in the web and we can expect to find a large share of web talking about the puppet compared to Chef and CFEngine, but when it comes to the helpful links that points to some thing interesting every product have similarity with CFEngine leading with a small margin.

4.1.3 Social discussion

Hackers news contains discussion on wide rage of topics, the data from this site was collected to get the discussion activity surrounding our interested products. Hacker news provides an API through which one can query it's database mentioning a set of search filter. A research was made for consuming the data from this site, and found a number of previously made solution by the community targeted to obtain a summarized and specific report out from the data exposed by api call. Hacker news Apps list a number of application that consumes data from hacker news api and gives users the flexibility to carry out research on those data by providing different functionality like viewing the trends, making correlation between trends for the supplied key words etc. Trend viewing tool was used to inspect the popularity of these tools. The application collects the keywords to be plotted as comma separated values and gives a time series graph. Internally the app will match the key words with words in different blog articles and discussion. Hence finally create a weighted mean of the frequencies of the words count found for each key word within a particular time range. Figure 4.3 gives the Correlation between the frequencies of key words CFEngine, chef and puppet cited in different different blogs and articles. It shows the puppet and chef to be popular and more actively discussed as well as there are more blogs and articles talking about the puppet and chef. But at the same time it also shows the artifacts being introduced in data as the graph shows a substantial traction in chef before chef was release i.e. in 2009.

4.2 Mailing list

The data collection process for analyzing the mailing list was diverse and driven by the of availability of mailing list data for public by these respective products. CFEngine provided it's mailing list archive data as mbox files that could be downloaded, Puppet was using google groups as it's mailing list hence it data was only available for viewing through web, finally chef also has exposed their mailing list archives as read only source. Hence for both Puppet and Chef the only way to get data was through crawling their mailing list.

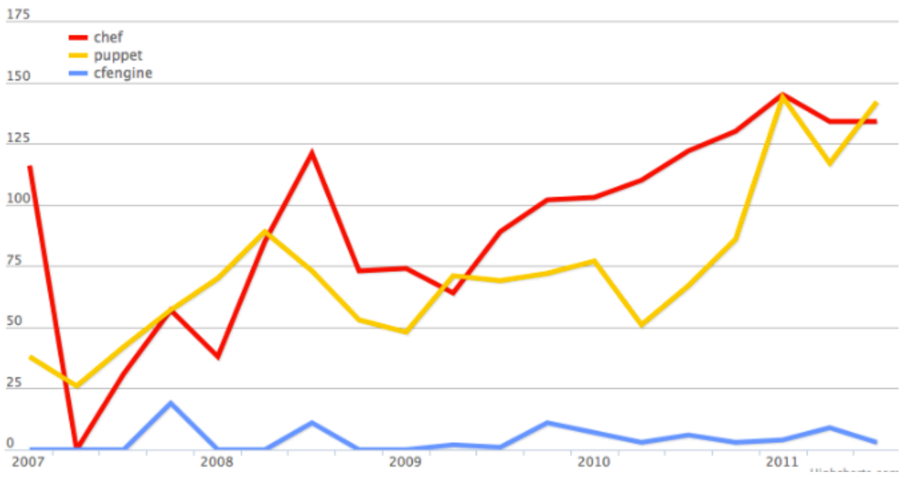


Figure 4.3: Articles and discussions from hacker news

Since only the users mailing list was to be analyzed , following are the mailing list that were taken into account

CFEngine	help-cfengine@cfengine.com
Chef	chef@lists.opscode.com
Puppet	puppet-users@googlegroups.com

CFEngine mailing list archive data was available for download for as mbox files. The mbox files were downloaded and the data was processed by the tool called mailingListStats , originally created as a part of set of tools that are used for analyzing open source project. This tool took the mbox files and parsed it to create a relational database containing the messages and people and the relation between them. A separation of threads , answers for the threads , senders , only question posters were carried out by making a SQl query against that database.

Puppet using google groups as mailing list did not provide any of it's archive data to be downloaded. Hence a crawler need to be written in order to obtain data from the puppet's mailing list. Since new user interface of google group utilized a a lot of javascript to render messages , it was not possible for crawler to get the message and it's details from google groups, But it turned out that puppet has it's mailing list data searchable through various free mailing list aggregation service. These services like mark mail and mail-archive also made puppet mailing list data available. Since they were simple html files , it was easier to parse them and grab the contents needed to from page and save it to database as web site was crawled. A through study about the placement of organization of messages and navigation mechanism available in these services for exploring the mailing list was done. It was discovered that mail-archive did not have a smooth navigation system that let the crawler script to gather all the mailing list data systematically. Hence mark mail was chosen, this site has two version of user interface , one targeting the human user, while another was friendlier for the crawler scripts to traverse the link and gather mailing list data. Finally a crawler was made utilizing perl language that was specifically designed to traverse the markmail links and gather data into mysql data base. Fields like date send , subject , message body , original thread name, sender, and it's url was stored. Utilizing these various Mysql queries were made in order to get desired out put like number of senders , separation of threads(questions) only from rest of the data etc.

Chef hosted it's mailing list archive utilizing the software called monharc, and it turned out that the mailing list archives data was only browse able from web by outside users. Thus a crawler needed to be developed in order to capture data from the web. But the challenge here was different compared to puppet. The data available for chef was not crawler friendly , i.e. they were designed to be difficult to traverse and harvest data. Hence with a little modification of previous crawler script , we could not gather data from chef. For data gathering a thorough understanding of navigation in chef mailing list archives was necessary. Once we reach the message page , we need to have a good understanding of html layout of the page to pick out only the things needed. Hence a completely new script was written utilizing perl to harvest data from chef mailing list. While the script seemed to successfully harvest data for some years, in others it failed due to invalid html and injection of script to render mailing address of the subscribers. Hence along the way various tweaks were made in the scripts to cope up the differences in the page and finally data till 2012 march 3 was obtained and save in the database. The fields extracted from mailing list and save in the data base was same as puppet and used the same set of queries to get our result.

Fig 4.4 shows the message activity collected for all of these products. It is a time series graph and shows the message flow per month since the year of the mailing list was established. The graph for CFEngine shows strong increase in mailing list activities at year 2010 while recently the activities has slowed down. For Chef , the mailing list activities are ever increasing. For puppet , it is observed that it has high initial value i.e.(messages per month) in graph , it is because Mark mail started to gather it's mailing list activities starting from that particular date. Puppet shows large numbers of mails being exchanged and steady increase followed by a slight deep in numbers in recent years.

With an idea of message flow in each of the mailing list we inspected the mailing list for seekers ,providers and both , The table4.1 summarizes the information obtained from each mailing list. The data for each mailing was collected from it's establishment to 2012 march. An interesting thing observed is that the numbers of individual falling in each group doesn't seemed that much different for all the products.

With a overview of mailing list data divided into different categories of concern, individual groups were focused and the data was further analyzed to study the activities performed by each group and understand importance of the group in over all mailing list. Data about metrics discussed in approach were collected for each group , except the metric "numbers of answers" was not collected for seekers group because, this group are solely categorized to identify the question posters and thread starters , thus they are not involved in any kind of answering activities. Table 4.2 shows the data collected for

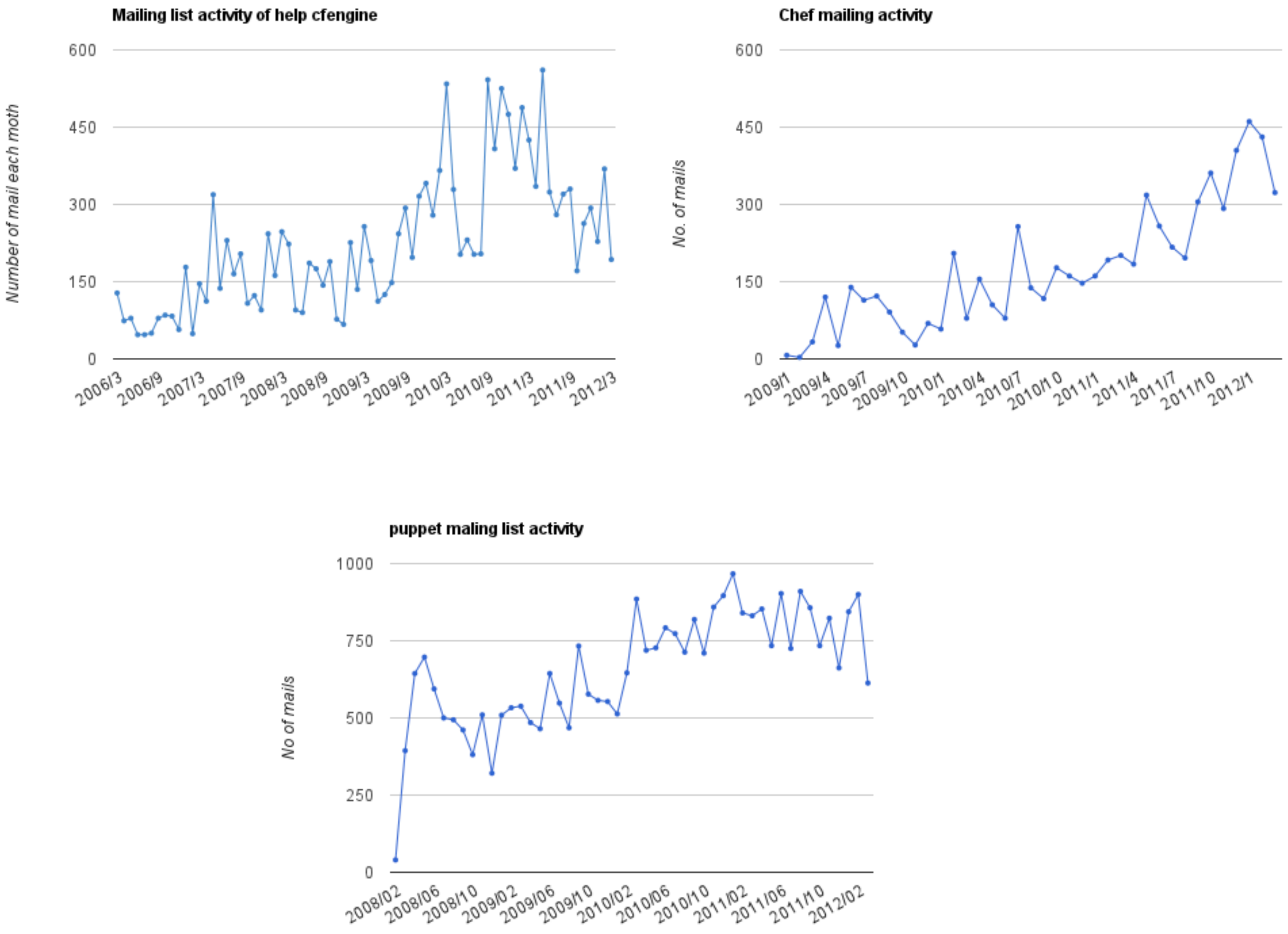


Figure 4.4: Mailing List activity for all there product

	CFEngine	Chef	Puppet
Seekers	33% *(n=269)*	19% *(n=110)*	26% *(n=834)*
Both	54% *(n=442)*	61% *(n=358)*	50% *(n=1570)*
Providers	13% *(n=112)*	20% *(n=112)*	24% *(n=739)*
Total subscribers	100% *(n=823)*	100% *(n=580)*	100% *(n=3143)*
Threads	*(n=3767)*	*(n=2056)*	*(n=7023)*
Resposes	*(n=12559)*	*(n=4745)*	*(n=25678)*
Total messages	*(n=16326)*	*(n=6801)*	*(n=32707)*

Table 4.1: Mailing list divided into different categories

	CFEngine	Chef	Puppet
Question posted	16% *(n=601)*	9% *(n=185)*	15% *(n=1023)*
Only one question poster	69% *(n=186)*	66% *(n=73)*	86% *(n=719)*
Unanswered threads	48% *(n=292)*	49% *(n=90)*	30% *(n=304)*
Questions replied on same day	39% *(n=233)*	38% *(n=71)*	45% *(n=467)*
Represents question poster	37%	23%	34%

Table 4.2: Statics about seekers group

	CFEngine	Chef	Puppet
Question posted	84% *(n=3165)*	91% *(n=1871)*	87% *(n=6180)*
Answer supplied	96% *(n=12143)*	94% *(n=4493)*	89% *(n=22959)*
Only two question poster	%35 *(n=186)*	13% *(n=358)*	19% *(n=301)*
Unanswered threads	22% *(n=292)*	30% *(n=578)*	16% *(n=1046)*
Questions replied on same day	57% *(n=233)*	52% *(n=966)*	61% *(n=3808)*
Represents question poster	63%	76%	66%
Represents answer provider	79%	76%	67%

Table 4.3: Statics about seekers-providers group

seekers group for all the tools. And table 4.3 shows the data collected for users that played the role of both seekers and providers. Here also one thing can be noticed immediately, All the products showing not much difference in the data collected for most of the categories.

Since data collected for both seekers and seekers providers, showed majority of queries and answers are made from this group , it was not necessary to collect data for providers , with simple arithmetic once can know the statistics for providers group for the different metric we have chosen. With an arsenal of data from mailing list for all of these products , we wanted to study the topics and problems discussed with in each products. The threads subject summarizes the message purpose of the thread and discussion happening inside it, a word count mechanism used in data mining was implemented only for analyzing the these subjects. The topics that are discussed in multiple thread should show up as one having high frequency. Since counting only one single word provides no meaningful results ,the word groups with in subject was counted. A per module to write a script that returns the count of bi grams , trigrams and quad grams with in the supplied text. For the analysis topics and problems discussed in mailing list , a query was run against MySQL data base to return only the subjects of the threads and the data miner script was run against the list of text generated. Top 10 frequently discussed topics found out by combination of two words, three word and four words ,in the mailing list of these products are shown in table. Of course a lot of interpretation can be made out from this result but it is to be analyzed to get a overview of topics and understand the problems frequently faced by user.

For generation of lists of topics frequently discussed , a set of commonly occurring words like re, release, released , available etc were excluded. The words representing the package release activities were also excluded. Also repetitive observation of result list generated by script was done to get a sensible meaning out of the words with in list. The combination of words that did not give any sense e.g. chef chef , puppet users ,cfengine cfengine were removed from the bi-gram list. The trigram list is to be use to make sense out of the bigrams and Quad-grams are to get a complete meaning out of bi-grams if any of the quad-grams contains combination from bi-grams. The table 4.4 shows filtered results from the miner script organized as bigrams , trigrams and quad-grams. Notice an interesting pattern is can be observed with in a table. The bigrams gives the view of topics frequently discussed while trigrams helps to create a mind set if the topics are from reoccurring problems and finally quad grams helps to see the topics found in bigrams are indeed problems. This gradual addition of words from bigram to quad-gram helps to comprehend result and perceive the problems more easily.

4.3 Bug Repository

4.3.1 Data gathering

According to the model of choice,failure and fault counts for a certain period of time was needed. Bug tracker was the perfect place to look into to obtain such kind of data. Bug tracker of these products stored the data about various bugs that appeared during testing as well as in production as bugs are reported by customers too, on top of that these bug reports also have version associated with it which will ease the task of separation of bugs according to version number. Hence focus was made in harvesting data from the bug tracker of these projects. Next step after identifying the source was identifying the attribute of a bug report that are important. A weekly bug frequencies for at least 3 versions was to be established, so at least three field were need.

- the date at which the bug was reported
- the version of product the bug is associated with
- unique bug identifier

The bug reports in bug tracker of all the three products provided various information. Additionally bug tracker of chef and puppet also maintained bug reports about other projects and components e.g. puppet enterprise, chef cook books etc. These bug reports has nothing to do with the main product so only the bugs related to project puppet as mentioned in Puppet bug tracker , and chef as mentioned in chef bug tracker was considered. The bug tracker of these product were deployed into service on various time hence shows various levels of maturity and the amounts of bugs posted in them. Therefor just for data

	CFEngine	Chef	Puppet
Bi-grams	cfengine help	run list	puppet dashboard
	insert lines	knife bootstrap	custom facts
	policy server	data bag	external nodes
	segmentation fault	remote file	best practices
	server returned	ec instance	ssh authorized
	returned error	client run	os x
	edit line	application cookbook	package provider
	duplicate selection	ruby block	undefined method
	package installation	attribute value	custom function
	quickstart guide	cookbook files	retrieve catalog
	package method	lwrp chef	external node
	package management	undefined method	authorized key
	file select	rhel packages	best practice
	reference manual	centos rhel	ruby dsl
	copying files	server error	puppet master
	syntax error	error chef	custom type
	config file	cookbooks chef	puppet labs
	cfengine stdlib	package provider	puppet agent
	var cfengine	nodes chef	file type
Tri-grams	files cfengine help	with chef chef	users how to
	file cfengine help	run list chef	users could not
	server returned error	on windows chef	could not find
	classes cfengine help	for chef chef	could not retrieve
	cf cfengine help	centos rhel packages	users problem with
	cfengine cfengine help	application cookbook chef	users problems with
	selection of value	how do i	not retrieve catalog
	duplicate selection of	attribute value in	users using puppet
	how to run	i test an	ssh authorized key
	error cfengine help	test an attribute	not working puppet
Quad-grams	duplicate selection of value	attribute value in an	could not retrieve catalog
	getting started with cfengine	do i test an	could not request certificate
	with cf cfengine help	an attribute value in	retrieve catalog from remote
	forget the domain name	test an attribute value	catalog from remote server
	package installation of rpms	i test an attribute	not retrieve catalog from
	did you forget the	how do i test	retrieve current state of
	you forget the domain	modifying kernel parameters chef	to retrieve current state
	the domain name or	value in an chef	failed to retrieve current
	started with cfengine webinar	undefined method ′ for	current state of resource
	domain name or ip	method ′ for nil	is it possible to

Table 4.4: Collection of frequently repeating word combination in mailing list

gathering process , all data about the bugs reports were collected from respective bug tracker. This collected data was filtered before analysis is done , which is the second phase. The bug tracker of CFEngine and Puppet allowed it's bug reports to be downloaded into csv files while chef's bug tracker allowed it's data to be downloaded as excel files. The data in all these files were imported in to mysql data base for getting the overview and then deciding on which filtering mechanism to be applied in each in order to obtain similar result format for all the products. The table 4.3.1 shows the source and the time of the bug tracker has been into operation for each product.

CFEngine	http://blog.cfengine.com/bugtracker	2010
Chef	http://tickets.opscode.com/browse/CHEF	2009
Puppet	http://projects.puppetlabs.com/issues	2008

4.3.2 Data Filtering

Each mysql table created by importing the data from csv and excel file contained different fields and large number of data than required. Hence data was inspected for each product by making SQl query to check the versions available , the date since operation etc. After carrying out inspection following data was to be filtered out for at least three version ,and the date range taken into consideration was influenced by the version taken into consideration.

For example for Puppet we could see a number of minor version but most importantly 4 major version i.e., 0.25, 2.6, 2.7, Telly. Since Telly was relatively new and it did not contain much bug reports associated with it three version were taken into consideration i.e. 0.25,2.6 and 2.7. Now the date range needed to accommodate bugs reports from all the version taken into consideration. For this case the oldest version into consideration is 0.25 ,whose initial bug report dates back to 2008 january. Hence the date range taken into consideration is 2008-01-01 till date.

Similar operation was carried out for CFEngine and Chef. Since CFEngine version 3.3 was just released it was not good candidate to measure fault count. Hence 3.0, 3.1 and 3.2 was choose for CFEngine and 1st bug report for the old version i.e. 3.0 into consideration dates back to 2010 january.

For chef there were number of minor releases for each version , but the major version were 0.6.x,0.7.x ,0.8.x 0.9.x and 0.10.x. The latest version 0.10. has number of minor releases i.e. 0.10.0, 0.10.2,0.10.4,0.10.6,0.10.8 and 0.10.10. Hence 0.10 shows it is a good candidate for analysis as it has been around for a while and has sufficient numbers of bugs affecting it. There for the version taken into consideration for chef was 0.8.x , 0.9.x and 0.10.x. The starting date taken into consideration for chef because of version 0.8 was 2009 june. The table 4.3.2 summarizes the date range and version considered

Product	Versions considered	Start date
CFEngine	3.0.x, 3.1.x, 3.2.x	2010-01-01
Chef	0.8.x, 0.9.x, 0.10.x	2009-06-01
Puppet	0.25.x, 2.6.x, 2.7.x	2008-01-01

4.3.3 Results

The Figures 4.5, 4.6 and 4.7 shows the weekly bug frequencies for all the three products separating the bug frequencies for each version out from the total bug frequencies. The outer line in the weekly plot shows the total bug frequencies. As seen in the figure there is a overlap in the bug reports between different versions which spikes up the over all frequencies. Also the over all bug frequencies patter doesn't seems to follow the pattern that can be represented by Weibull distribution function but the bug frequencies related to individual version seems to follow the pattern that can be represented by Weibull distribution function. Hence in analysis section , curve fitting can be carried out for each version to calculate the parameter for distribution and ultimately calculate reliability. It is also observed from figure 4.7 that there are small releases of bugs reported for some version of puppet early on from development phase and there is significant over lap of old version with new version that shows either the development of these versions are carried out in parallel or the entry in bug tracker are wrong. While for Chef and CFEngine in figures 4.5 and 4.6 respectively, bugs report arrival pattern seems to be more natural with least overlap on bugs reported for old and new versions and bug reports for new versions arriving only after the old version. Additionally when bug reports for new version starts to arrive, the bug reports for old version seems to die, It might be because of old version being phased out by users as new versions are launched.

4.4 Usability test

A usability test was conducted as planned in the approach section. The primary objective of this task was to collect the data about usability metrics that enabled us to make a quantifiable measurement regarding usability and give us a better view of the problems associated with it on these different products. The test executed as planned but there were few glitches that prevent test from smoothly moving forward. Since users were taking test at different time at their comfortable schedule, my of the users happen to get the same problem from the test environment i.e. restoring their respective machines when they are done conducting test with one products. Thus there was minimum email exchange between candidates during the test phase in order to guide them to properly use the test environment which seemed to take away some enthusiasm in candidate about the test. Apart from that users did not have any trouble understanding the test question and the process of submitting their response. The section below shows the results obtained for various

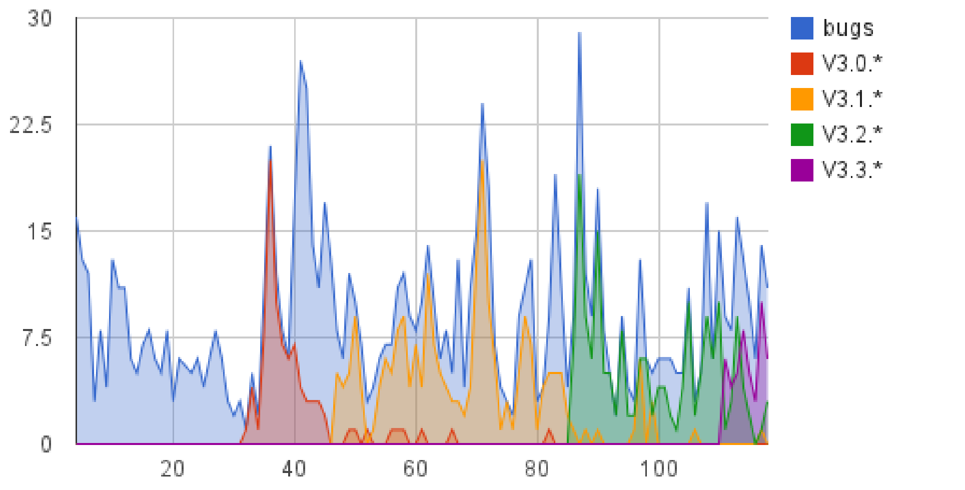


Figure 4.5: weekly bugs frequency for CFEngine

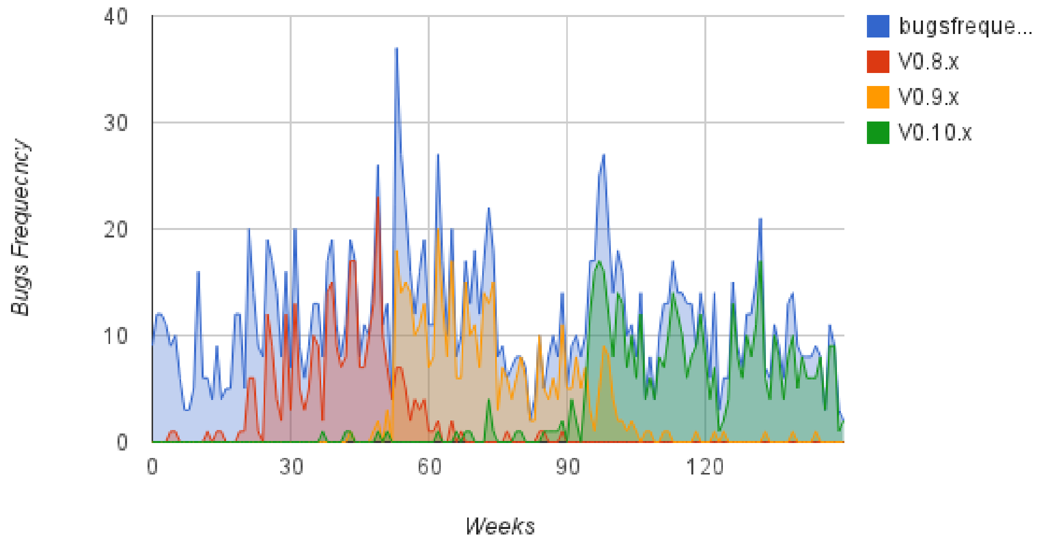


Figure 4.6: weekly bugs frequency for Chef

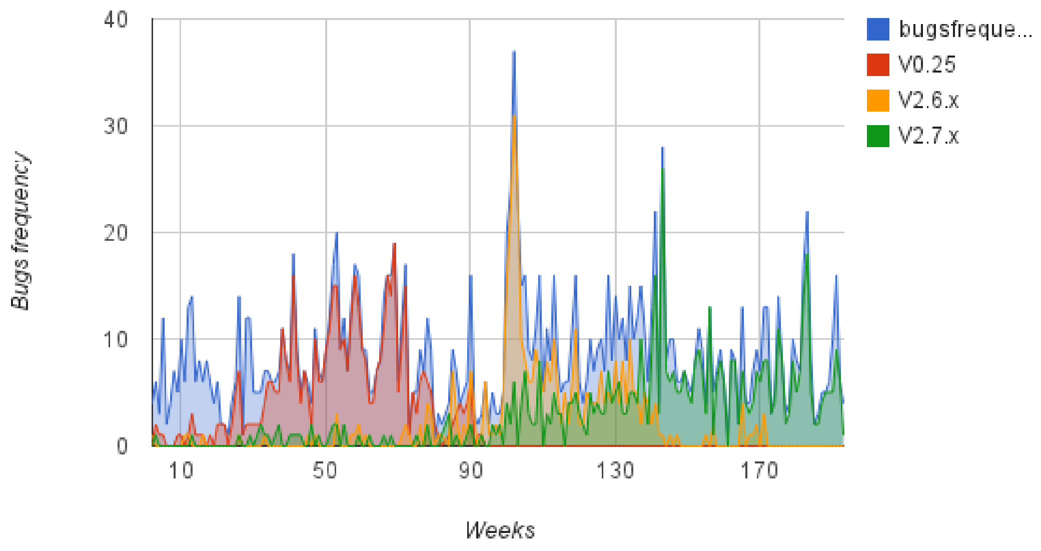


Figure 4.7: weekly bugs frequency for Puppet

usability metrics collected from test.

4.4.1 Completion rate and Task times

Task completion status along with task time for 5 candidates conducting 3 task on each product is shown in table 4.5. The entries in table with task time are completed and the ones without are incomplete task. The table shows a majority of users being able to carry out at least two task for CFEngine , while for Chef and Puppet equal portion of users being able to complete the tasks. But this collected task statistics is to be processed further by application of point estimation and calculation of confidence interval in order to get a meaningful interpretation out of it.

4.4.2 Task difficulty level

Task difficulty level was meant to be tracked by single was ease question. The table 4.6 shows the single ease question score for each task for CFEngine, Chef and puppet. The results shows that some most of the users have high level of task satisfaction for two task i.e task 1 and task 2 using CFEngine. For Chef there is mixed level of satisfaction regarding all the task. In case of Puppet majority of users is satisfied with their performance in task2.

Candidate No.	Tasks	CFEngine	Chef	Puppet
1	1	24min *(15min)*	22min	32min
	2	5min	9min	20min
	3	30min*(15min)*	41min	-
2	1	15min	27min	-
	2	11min	4 min	-
	3	28min	7min	-
3	1	15min*((5min)*	-	-
	2	9min*(5 min)*	-	10min
	3	50min*(30 min)*	-	15min*(5min)*
4	1	45min	90min	-
	2	90mins	-	-
	3	-	-	-
5	1	45min	60min	10min
	2	-	-	35min
	3	-	-	20min

Table 4.5: Task times(unproductive time) for performing tasks

Tasks	CFEngine	Chef	Puppet
1	5	1	4
	6	5	1
	6	1	1
	3	3	3
	6	1	6
2	7	6	6
	7	7	1
	5	-	6
	5	2	1
	1	1	6
3	4	2	1
	3	7	-
	3	-	6
	-	-	-
	1	1	4

Table 4.6: Task scores supplied by users

Problems	**User 1**	**User 2**	**User 3**	**User4**	**User 5**
CFEngine Task1					
Bootstrap problem	x	x	-	-	-
dependency issues	x	-	x	-	-
CFEngine Task2					
Problem writing poilices	-	-	-	x	x
Problem understanding process flow	-	-	-	x	-
CFEngine Task3					
Lack of good examples and documentation	-	x	x	-	-
Chef Task1					
problem with ruby	x	-	-	-	-
problem with keys	-	x	-	-	-
cannot follow documentation	-	-	x	-	x
Time consuming	-	-	-	-	x
Chef Task2					
confusing terminology and architecture	x	-	x	-	x
Puppet Task1					
Certificate Error due to DNS	x	x	x	x	x
Puppet Task 2					
Problem understanding architecture	x	-	-	-	-

Table 4.7: Problems faced by users

4.4.3 Usability Problems

The problems faced by users while carrying out the task is termed as usability problem. These problems can be common problems that are faced by common users of product or it can be an unique problem only face by a specific users, what ever is the case if a problem is found it helps to understand the task metrics that users supplied after performing the task. Tables 4.7 shows the problems reported by users while performing tasks utilizing different products at a time. It cannot be related with whole scope of problem associated with product, it is only the problem reported by users while using particular product. This problems can be treated as obvious problem because if it is experienced in such a small sample , then it is likely to be experienced by a large set of users in large population. Most importantly the problem for Puppet seemed to be experience by almost all the users hence it must be the most frequent problem that users have to face while using puppet.

4.4.4 SUS scores

Post test questionnaire responses were collected as user's perceived satisfaction of the product itself. The collected responses were a series of scores ranging from 1-5 for 10 question asked to each participants about the product.

Users	CFEngine	Chef	Puppet
user 1	72.5	15	37.5
user 2	50	70	
user 3	65	30	62.5
user 4	47.5	2.5	95
user 5	57.5	35	15

Table 4.8: Calculated SUS scores from users for CFEngine , Chef and Puppet

These raw scores need to be converted into SUS scores(as single number) for the product by the users in order to proceed with subjective usability analysis of the product. The process to convert these raw data into SUS scored is described in the background section 2.6.4. Following that process each scores were converted into a range of 0-4 and the sum of scores was multiplied for each product obtained from a particular user was then multiplied by 2.5 to get corresponding SUS score. The table shows the SUS scores obtained from the 5 users for CEFngine, Chef and Puppet. The results shows that CFEngine has SUS score in rage of 50-70 while for Chef and Puppet the scores highly vary from users to users.

Chapter 5

Analysis

5.1 Community trends

It might me tempting to believe from figure 4.1 that chef an puppet are the technologies that are currently most popular and widely used but the data google insight result doesn't provide the full picture. Though it shows the -ve trend for CFEngine , +ve trend for Puppet and Chef it might be coupled with noise. Considering the fact that both puppet and chef are english dictionary word and the key word like chef and puppet could have been used for searches that has nothing to do with configuration management but have a huge effect in trend analysis. So keeping that in mind, even though operating system as filter was applied to signify the context of search the data seemed to be coupled with noise. There is significant level of activity for chef in year 2007/2008 while the project mailing list contains data only from 2009. The chef project actually started in year 2009. In year 2009 the searched frequencies seems to pick up a little. Also if the same can be observed for puppet, the time series shows the search terms for puppet started from early 2006 while the project actually started from 2005. If CFEngine is now taken into concern then it shows a high rate of searches in year 2005 / 2006 which is possibly for CFEngine 2, but CFEngine 2 popularity slowly died while CFEngine 3 was introduced in year 2009 there seems no clear possible way to get CFEngine 3 specific data. Hence it's trend is coupled with CFEngine2 which can be considered as a kind of noise. Also another important aspect observed from the google insight, is the area interested in these products. While CFEngine seems to be the global player where people around the Europe and USA being interested , it might be reflected by the used of CFEngine 2 which was the most popular configuration management solution at the time it released. But with puppet and chef as well as CFEngine3 being introduced lately and more younger they did not have enough data to represent the areas that were interested in them. Hence the area of interest gives the tentative regional use case of these products But none the less the time series 4.1 give a overview of what is the current status of these tool and how frequently are they being searched in the web. It seems like all of them have similar search trend which can be implied that all of these tool are equally used by the people. It might be some time when one of these tool leads the path or might remain same sharing the equal portion from the market between them.

5.1.1 Popularity and Resources available

It was the same case we faced with google insights however the the quantity of data returned from hacker news is much less and focused. Since it returns analyzing a set of articles submitted and the discussion under them , it's much more easier to filter the randomness in the data and try to filter out the results. With a small amount of filtering done by applying the key words like "kitchen food" etc the total data returned for chef seemed to reduce. Hence and statistical method like sampling[42] and testing [43] was utilized to predict what portion of the total data returned contained the exact result.

Small samples containing 10 results were taken repetitively from the total data returned from the search query. Then output obtained in each samples were analyzed to point out calculated the number of desired results. In total 10 samples were taken at random that included 100 results from the total result set. The number of desired results obtained in each samples are

9,5,7,4,5,7,2,6,8,6

Thus the mean numbers of interested results were 5.9. Hence from the sample collected , it showed that only 59% of the total result obtained for chef represent the data related to configuration management and system administration and useful to us. A confidence interval was calculated in order to find the interval that could represent the portion of total result set to contain the results we are interested in. A 99% confidence interval was chosen which resulted in the interval of (0.59-0.12, 0.59+0.12). There fore we now know from the samples collected that 47% -72% percent of the data result hope to contain the result about Chef related to configuration management.

After the result obtained, a confirmation was needed that confirms the results showed by the sample actually represents the total proportion of the population i.e.(the whole result) set. For this one sampled z-test was conducted. For z-test , a significance level of 0.05 was chosen. Since the interval of the proportion was known ,a hypothesis was made that 60% of the whole result set represented interested data, rest were noise.

Thus the values for hypothetical proportion of population(P) is 0.60 and sample proportion was (p) is 0.59. The z-test was conducted by utilizing R software. The p value obtained was 0.419 which is greater that the significance value. Hence the null hypothesis cannot be rejected i.e. our assumption that 60% of the whole data returned for the search query on key word "chef" contains the data about Chef product.

Therefore the total results obtained for chef was 1532 of which only 60% i.e. 919 of them were useful. This can be interpreted as the tentative total number of discussion, articles and blog post we can find for chef is 919. Similarly the result set for puppet keyword was analyzed to reflect the true number of post talking about puppet in IT or system administration. The random samples resulted following number of right results.

6,10,7,8,7,4,5,5,2,8

Therefore from the samples it shows 62% of data available is related to puppet and other are related to other things. 99% confidence interval for showed that data portion about Puppet product is in range 50% to 74%. Hence a confirmation test was made taking a hypothetical value 70%, after z-test , the p value was found to be 0.04 which is less than the significant value. Hence our assumption was not correct. Now a new assumption with 69% was made. For this hypothetical value the z test resulted, a p value of 0.06 which is slightly greater than significance value. Thus with 99% confidence, it can be stated that 69% of the whole result set contains the data about Puppet product.

Thus the total result returned for puppet key word by hacker news was 1202. Out of these results only 69% of 1202 i.e. 829 contains the discussion , articles and blog post about puppet as configuration management product.

For CFEngine only 83 results were obtained hence the methods applied for the Puppet and Chef cannot be applied to it , because total population size won't be greater than 10 times the samples size. Since the total population size was very small, every result were analyzed. After having a inspection of data all the data returned from the result set seemed to contain data about CFEngine as configuration management product. Hence all 81 records were useful. Thus the total resources as obtained from hackers news is listed in the following table.

CFEngine	81
Chef	919
Puppet	829

From the result obtained, it was apparent that as raw results includes randomness , so a statical method was applied to estimate the total proportion of the whole data that represents the useful information. 99% confidence interval was used to find the required interval and it was followed by one sampled z-test to test the hypothetical value of proportion taken from the previously calculated interval range. This method might not be perfect and accurately identify the true proportion of the real data in the whole result set , but it gives the possible view of interval that represents the proportion of desired data hidden in the result set. This method can be improved and made more accurate by taking large number of sample with different sizes repetitively and finally conduct the same kind of test on the data obtained from these samples. With the high numbers of samples it is more likely to accurately predict the true portion of data in the result set.

From the hacker news data analysis , our hypothesis about the result set obtained for chef and puppet will be influenced by a lot of noise and that with CFEngine will be much less was proved. It absolutely makes sense because CFEngine being a non dictionary word not used in any other field and profession will always give the result that we are looking for. While puppet and chef both being a dictionary word and linked with other profession (entertainment, cooking) etc tend to give a lot of result of which only some portion of data are useful to us but still the results point out that Puppet and Chef are popular and large numbers of resources are likely available for them.

5.2 Community Structure

5.2.1 Analysis from mailing trends

Data on mailing list was categorized into various areas of interest and we found a of similarity between these products on community structure. Though different amounts of mailing traffic and trends exits in each of the of the products , the structure is the same. Since puppet has large number of subscribers ,currently it has double the mailing traffic compared to other products, be chef seems to be gaining a lot of popularity with ever increasing mails since it's launch. For CFEngine , the mailing list traffic is a bit turbulent , with large numbers of mails arriving on the year 2010, it may be due to the launch of CFEngine 3 earlier in the year 2009. Hence it can be though as CFEngine is slowly making it's way back into this field with CFEngine3.

From the numbers of subscribers in the mailing list Puppet proves to be the popular product. Chef also proves to be the hot product with a bright future with as we can find a lot of resources talking about it provided that it is is just two years old, and +ve trend observed in it's mailing list also adds to it. For CFEngine it's mailing list shows to be the oldest of all , and the mailing traffic shows that it has small community that is involved in discussion and problem solving. When looked at the type of user heavily involved in questioning and answering all product showed the same groups of user were significant. All product showed seekers only and seekers/provides group to be very vital to it's community and their portion with in over all subscriber were similar in all of these tools. More than half of the subscribers fell into these category for all product.

5.2.2 Analysis of seekers group

CFEngine shows it has a large share of seekers only group that comes to the mailing list only to get answers , this remains some what similar to the puppet and chef. This can explain why a CFEngine has a lot of users but small mailing activity foot prints. Also the providers only group is same for Puppet and Chef while that of CFEngine is small. But important result figured out from this analysis was all of the product seems to have large share of users involved in querying and posting answers.

Moving deeper in to analysis of behavior and importance of each group, we can see that same share of question are posted by the seekers only group for puppet and CFEngine despite their size. They represent 1/3 of the question poster and post almost 1/4 of the total questions. This can be understood that large share of people use it's mailing list for help. It is also seen that a lot of queries from these group get unanswered for all products. Chef seemed to be leading into this category while CFEngine is very close. This might be due to the users in this group post already discussed problems or a lot of messages in from this group is spam. But if it is a normal user ,the numbers in this section should be low as possible, then only it encourages the community to flourish.

Puppet seemed to be more responsive when it comes to answering the queries from seekers group but only by small margin. Combined with the queries dropped and the responsiveness , it can explain why puppet is the popular product among the three for novice users. But in summary all of the product tending to have similar behavior in number of categories and differed only by small margins.

5.2.3 Analysis of seekers providers group

All the product showed that a lot of it's mailing list users belong to this category, and majority of question and answers in mailing list are provided by this group. But even with this similarity as a whole , small differences can be observed when each of the individual categories focused. This group provide large number of question compared to other two groups but only with small margin. Similarly , for CFEngine this group provides most answers as compared to it's counter parts. The portion of questions that went unanswered from the user in this group was less for puppet and high for chef, but in all it was less compared to that of seekers group. Also the portion of queries from the user in this group that get answered in first day is large for puppet and low for Chef but the differences being very small. More than half of the question get replied on the first day. Combining the response rate and queries dropped we can see that community is more supportive and helpful. Hence when one becomes a experience user of the products , one can often look at the community to get problem solved / discussed for all of the products.

But there was one phenomena seemed similar in these products. Even though from top, this group seemed to be responsible for providing answers to the queries, there was only a small set of users responding to lot of the threads, i.e. answering queries, And lot of other users were involved in answering small portion of the queries. Thus with in this group also lie a subset of users that can be termed as power users and knows a lot about the product. This gave us a new insight in to community structure and support they provide for these product which is further discussed in 5.2.4.

5.2.4 Load distribution

A simple plot of the numbers of responses and responders indicated that distribution of response to responders is seemed to follow a power law [44]. Roughly it is estimated that for all the product community member's response pattern can follows a power law ,called 80 -20 [45] rule, which laid out in simple terms says 80 percent of the total answers are given by 20 percent of the users. This might lead to the conclusion that size of the community is one thing but there are only some core and power users that keep the community supportive and alive and this is same for all the product. To get a better understanding of the phenomena and reach to a conclusion , the data set with responses and number of responders associated with them was to be quantified by a mathematical model. Additionally a mathematical model will help us to understand the load distribution in the community it self. Finally with the response to responders distribution we can also have a good understanding of knowledge distribution and support offered by community.

To quantify it mathematically power law was taken as theorotical model, our assumption of it followed 80-20 rule as known as Pareto Principle [45] can be examined by plotting a Lorenz curve [46]. For the distribution to follow Pareto Principle the Pareto index α should be approximately 1.161. This index can be calculated with the help of Lorenz curve and Gini coefficient with equation 5.1.

$$\frac{1}{2\alpha - 1} \tag{5.1}$$

The Lorenz curve is used in economics to find the wealth distribution and inequality on wealth concentration within individuals in society. It also suits for this case as well for finding out the concentration of responses with each individuals and ultimately study the dependence of the community support on these individuals. The Lorenz curve shows the inequality in terms of Gini coefficient. In Lorenz plot the perfect equality is shown by the diagonal line (0,0) to (1,1) in figure 5.1. In this case perfect equality shows that every community member is equally involved in discussion and answering. Well it is not possible in real world, however if majority of community member is active the our lorenz curve plot must be near to this equality line, i.e. the area between curve and the line which is gini coefficient should be small. Hence higher Gini coefficient value signifies larger is the inequality and majority of community is less involved. The perfect inequality is given by $y = 0$ when $x < 100\%$ and $y = 100\%$ when $x = 100\%$, simply said all the rest of community member is passive and only one is active in answering. This will result in perfect "L" shaped curve. The more curve looks like "L" the worse is the participation of community members and greater is inequality. For this case, high inequality shows that dispersion of knowledge is very little and only few people is able to answer the majority of queries. This inequality also signifies that community support offered by this products is dependent on few individual. Thus an attempt was made to categorize which of these product has hight inequality and by how much.

From the calculated gini coefficient, even if the Pareto index α is not 1.161, we can find the ratio of concentration i.e inequality, It can be calculated with the following relation 5.2 where H is gini coefficient.

$$A : B = (\frac{1+H}{2}) : (\frac{1-H}{2}) \tag{5.2}$$

The lorenz curve plot, along with the gini index for the three products is shown in the figures 5.1, 5.2 and 5.3. After the gini coefficient was obtained, it was apparent that 80-20 rule doesn't hold true for these products , the Pareto index α calculated was more than 1.161 which showed that there was even more in equality i.e. 90-10 or 95-5. Hence equation 5.2 was use to find the inequality. Table 5.2.4 summarizes the joint ratio (degree of imbalance) observed in three products with the obtained gini index.

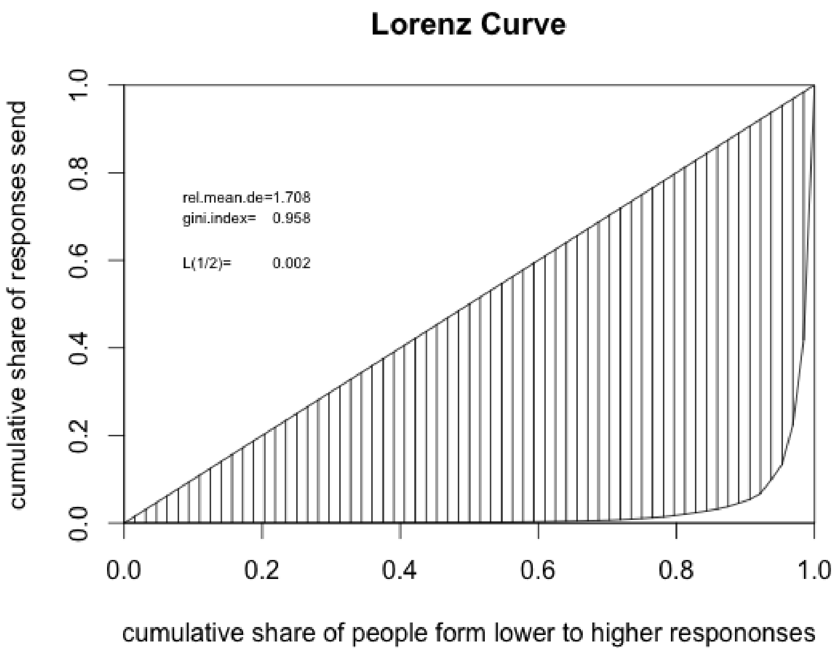


Figure 5.1: Lorenz curve plot for CFEngine

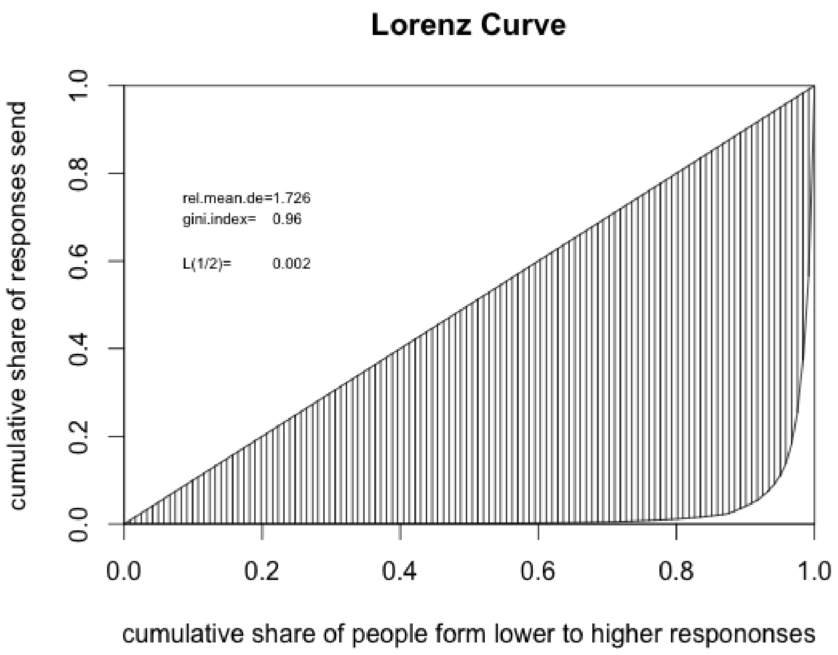


Figure 5.2: Lorenz curve plot for Puppet

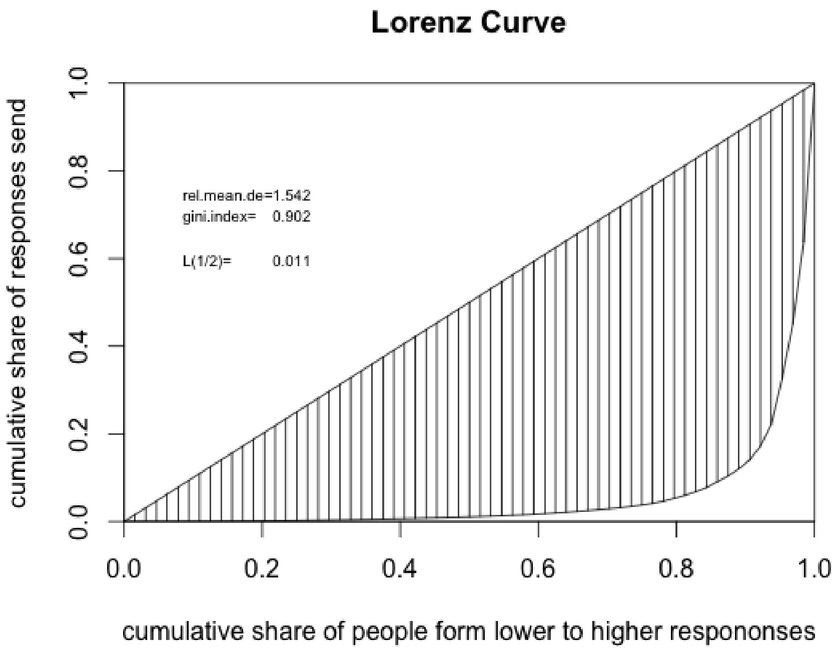


Figure 5.3: Lorenz curve plot for Chef

Product Versions	Gini Index	Joint ratio
CFEngine	0.958	98:2
Chef	0.90	95:5
Puppet	0.96	98:2

This gives a clear description that all the products have high amount of in equality i.e. very imbalanced when it comes community member participating in discussion and responding to queries. Despite the size of community and popularity shown by the Puppet ,it's joint ratio is same as the joint ratio of CFEngine , i.e. 98% of response is from 2% of people, it shows that community size doesn't have any effect on the support you get from mailing list. And one always have to depend in these very 2% user to get support. This also shows that these two percent of users are have much knowledge about the product and are actively sharing it. Chef on the other hand being a new product , shows some what different behavior. Though the joint ratio for it is also severely imbalanced, it's some what better than CFEngine and Puppet. It might be because it is a new product and more users are novice to it so every body is taking their time to response and take part in discussion. Down the time it might as well take the path of CFEngine and Puppet as old users acquire more knowledge and experience with product , so they have all the knowledge and only they are answering the queries and discussing while other intermediate and novice users just becomes a passive users.

5.2.5 Analysis of Data miner script output

The table 4.4 shows the top 20 frequently occurring bi-grams , top 10 trigrams and , top 10 quad-grams. When the list of bi- grams is inspected , mix of list and topics that are discussed in the mailing list can be observed. Tri grams gives a clue what topic is about and the looking at the quad gram the problem associated with the product can be pointed out.

CFEngine

From bi-gram inspection of CFEngine mailing list it can be said that people are mostly posting issues related to package management and editing files. It also shows that the users are having trouble finding the quick start guide as well as there seems to be discussion around CFEngine reference manual and standard library. Possibly these discussion can be made due to user not finding the information that he/she is looking for in reference manual or improving the Reference manual to make it more understandable. Also CFEngine standard library contains a set of reusable body that can be utilized in CFEngine policy files , it appearance in the list can be summed up as people wanted to add more things on it or not being able to use the functionality within it. Hence looking at just the bi-gram list an overview of the possible problem we might encounter can be observed, it is also helpful to have a idea what topics are being brought up frequently.

When tri-grams are observed, it shows that people are having problems with implementation of classes in policy files also making files related policy task. CFEngine also showed problem with selection of value.

With the Quad-gram inspection, the problems that people are facing becomes quite clear. It can be assured that users are definitely having trouble with package management and they are searching for the easier way of learning the CFEngine, which adds to it's reputation of being a hard language to learn. Also we see users are experiencing problems of domains with CFEngine, but this was error associated with CFEngine 2 which was no longer relevant.

Chef

The bi-gram list of chef shows chef specific words like data bags , run list , knife boot strap etc as frequently stated words in the mailing list. This signifies that a lot of problems and discussion are taking place on these topics. EC2 showing up in the link shows chef users are using it with cloud services. Also users of chef are looking for ready made cook book files. Since cook books are the blueprint of configuration used in Chef, we can understand people are looking for different cook book files and usually turning up to the mailing list to find cook book files that fulfills their need. This also shows the cookbooks in hosted chef are difficult to use in users context or doesn't cover the whole requirement of users. Another interesting topic that showed up was attribute value , these attribute value are dynamically calculated by ohai plugins and also definable by user through json. It shows users are having problem with implementation of attribute value.

With a closer inspection of to 10 tri-grams list it was found that there seems lot of talking going on about the implementation of chef on Windows and Redhat packages. It might be due to chef has recently released windows package and users are having trouble implementing it, appearance of Centos Redhat package signifies either the user cannot find the required package or there is some flaw with in that package of the chef. Another interesting thing observed is that user are having trouble testing the value of attribute.

From the Quad gram analysis, it absolutely clear that users were definitely having trouble in testing the attribute value. It can be assured by the lack of dry mode available in chef to test the configuration rules and also since it is possible to declare attributes at several place like cookbooks, nodes, roles ,environment and since attributes are just the key value pair, the values may be over written. Thus there is an strong used case of testing an attribute value but it seems people are finding themselves in trouble trying to test it.

Puppet

The bigram list of puppet showed that discussion are happening in the topics like custom facts, osx, secured authorization, best practices etc. From the topics it seems likes users are struggling to implement custom type, external node and more importantly retrieving catalog. Having a look at the list it can be roughly estimated that users are looking for the way to define custom facts because the available facts didn't cover their whole use case. There are problems related with authorization either due to human error or because of puppet itself. Puppet master authorizes the client based in the ssh key exchange between client and master, but it is observed that people are having trouble implementing it. There also seems to be some frequent discussion related to best practices. People always looks for best practices , to make things modularize and reusable when things grow. The topic "best practice/s" shown in the list can be understood as it depends on our implementation to make things organized and reusable with Puppet. Lastly a glimpse of puppet master failing to give catalog(configuration specifications) to it's client is seen but this can be further confirmed with tri-gram and quad gram analysis.

With tri-gram analysis we can see that there main topics /problems are dominant in puppet mailing list. The problem related with ssh authorization , problems in retrieving catalogue and users searching for the best practices of doing things.

From quad-gram lists it becomes absolutely clear that puppet has many cases where it failed to deliver the catalogue to the clients. Catalogue is the set of configuration list computed by puppet master for a particular client. A catalogue cannot be delivered when a server is too busy to respond to the client or when there is problems in authorization between client and server. It is also seen that the problem of authorization occurs due to failure to receive certificates to the from the server. Another major problems face by puppet users , is the inability to the retrieve the current state of the system and resource. This retrieval of current state is to be done in order to find if there is any deviation on the system and resource than the one mentioned in catalogues. Therefore the quad grams lists shows that puppet suffers from two main draw backs , that can potentially make it unreliable ,one is it fails to retrieve fails from server, while another is if even configuration files successfully are retrieved, it might fail during implementation due to error associated with retrieval of current state of resource.

5.3 Reliability

5.3.1 Distribution Fitting

The figures 4.5, 4.6 and 4.7 all shows that weekly bug frequencies related to each version follows a pattern that can be represented by Weibull distribution function. Utilizing R's maximum likelihood estimation(MLE) technique the parameters for Weibull distribution was estimated. Since R requires time domain data, relative frequency of bug reports needs to be converted to occurrence times of failure. Therefore bug report is mapped to it's corresponding weekly period. For example 2 bug reports in week 5 and 4 bug reports in week 6 is now converted to 5,5,6,6,6,6. Thus the input provided to the R for estimation of parameter is cumulative which consists of list of bugs with each bug converted as week number. A histogram of bugs frequency is plotted for each version with x axis representing the weekly time and y axis representing the number of bugs reported in that week. Using the MASS package available in R , Weibull PDF was fitted in the bug frequencies histogram plot [47]. The output of the curve fitting method will yield scale α and β shape parameters which will be used to calculate the reliability of product weekly. Measure the goodness of fit can be carried out using relative measures. The equation used to measure the goodness of fit is given by equation 5.3. The index expresses a coefficient of variation of root mean square deviation normalized to it's mean measure of the mean of observed values.

$$\delta = \frac{\sqrt{(}\sum_{i=1}^{n}(y_i - {y_i}^*)^2)}{\sum_{i=1}^{n} y_i} \tag{5.3}$$

The computed scale and shape for each version of the three products along with the accuracy level is shown in the table 5.3.1. As discussed in the background theory , the effect of scale parameter is to squeeze or stretch the PDF plot. Greater the value, greater is the stretching, which produces the flatter curve and this implies less rate of failure. The variation level is the match between the fitted curve data and the actual data i.e. bugs density. Less the value of variation between the observed and fitted value, the accurate the match is.

Product Versions	**scale**	**shape**	**RMSD**
CFEngine v3.0	43.257	4.278	86.36
CFEngine v3.1	73.313	5.333	63.27
CFEngine v3.2	103.373	11.120	69
Chef v0.8	45.302	3.677	48
Chef v0.9.x	79.669	4.48	56.91
Chef v0.10.x	124.157	7.088	50.80
Puppet v0.25.x	60.89	3.49	44.56
Puppet v2.6.x	118.670	5.095	65.295
Puppet v2.7.x	155.97	4.987	43.272

The figures 5.4 , 5.5 and 5.6 shows the Weibull distribution curve fitted with the bug frequency for each versions of the three products. The parameters listed in the table 5.3.1 was used for corresponding versions to get the plots. From the figure , it's visually clear that puppet has the better fit , much accurate compared to others, this also is supported by the RMSD value in the table 5.3.1. The increase and span of failures are correspondent to shape and scale parameters respectively. The estimates are favored towards scale to cover large time span. Thus this pushes down the peak of the fitted graph resulting a low accuracy level specially if there is high amount of failures in small interval of time.

This explains the accuracy obtained for CFEngine for most of it's version and it is also observable in figure5.4. For each point(week) in horizontal axis, difference between observed frequencies and expected frequencies from Weibull parameters is high which increases the over all sum of the residuals. Therefore the calculated ratio of sum of these differences with mean of observed frequencies, will result a high coefficient of variation of theoretical frequencies about mean of observed frequencies. The amount of error of the distribution fitting is also clearly observable in the graphs. These kind of errors could have been made more transparent by obtaining the residual matrix i.e. that contains difference between the observed frequencies and expected frequencies and plotting them against time. If residual matrix was plotted then , it would also have offered a visual clue for the reason of such a large variation in some plots.

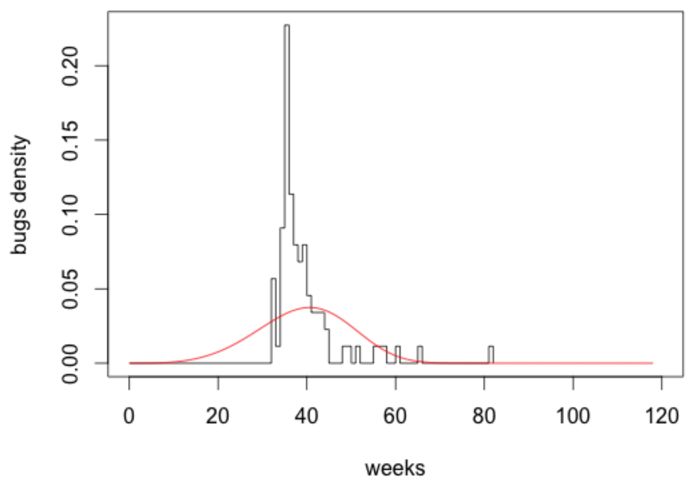
Another method will be to calculate accuracy on basis of pattern rather than magnitude, is to calculate the correlation between the observed frequencies and frequencies obtained. Correlation can explain which versions of bug reported follows the pattern shown by the fitted curve more closely. Therefore, if there is positive correlation then the obtained frequency distribution follows the theoretical model generated from the parameters using MLE. Nearer the correlation coefficient to 1, strong is the correlation between bugs frequencies observed and fitted ones.

In this case the important aspect is the pattern of bug report as peak of estimates are factor of time span rather than magnitude of failure rates. Therefore as long as estimates are increasing or decreasing with respect to observed failure rate , the model is able to describe the failure behavior. It's reasonably clear that all the fitted frequencies demonstrates the +ve correlation and majority of magnitude variation in between observed and fitted ones are same for all the product. For demonstration the correlation between fitted and observed bug frequencies was calculated and it was find to be 0.71 which is a good indication of estimated PDF is good representation of failure behavior. So the data obtained were considered for calculation of reliability for each versions with in the product.

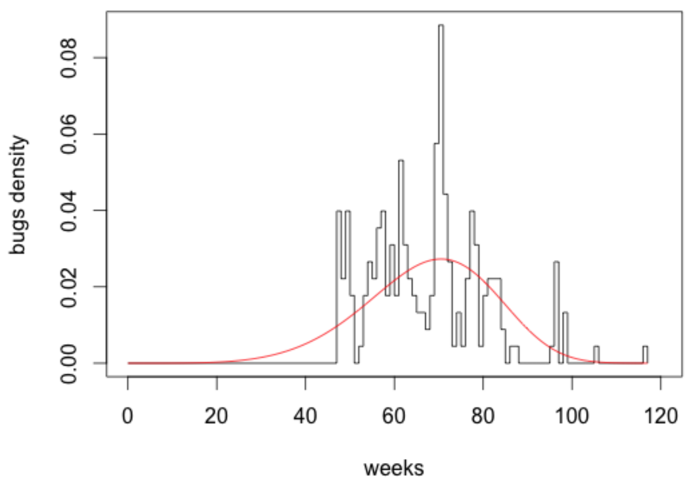
From the figures it's clear that bugs in all the version of these product follows the Weibull distribution showing increase in bug numbers being reported on early weeks and stabilizes slowly later on. A similar trend in bug pattern for CFEngine and Chef from fig 4.5 4.6 i.e. minimal overlap on span of time for bugs reported for consecutive versions. In puppet there is a significant portion of time span overlapped between two versions. Hence since the bug in each version is spread over some span of time and there is no sudden spike in the bug numbers reported in each week , we have a better fit of model for puppet version compared to CFEngine and Chef.

5.3.2 Reliability growth

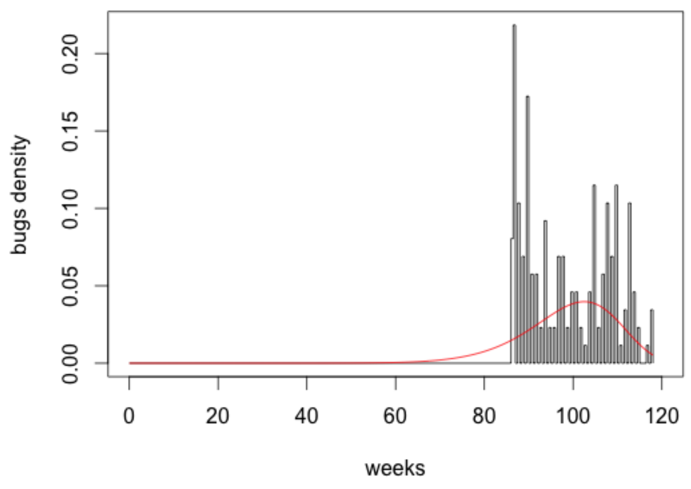
Since there are two aspects to look into the plots i.e. numbers of failures which is frequencies of bugs(shape) and the span of failures (scale), i.e. the time span where the bugs reports about the particular version continue to show up. Estimation favors the scale to favor long span of time because it is natural for bugs to show up for some period , but if it shows up for a long period of time the it is surely unreliable product. The Weibull's parameters value calculated for each version was used to calculate the reliability of each version of the product utilizing function 3.1. Figure 5.4d,5.5d and 5.6d shows the reliability graphs for each products with various versions. Since the time range considered for each product is different, comparison of reliability of versions was carried out within the product only. As expected the new versions offers more reliability in all the products. In case of reliability CFEngine shows to be a better product that offers more reliability on each new version shown in 5.4d, while Puppet seems to have a small amount of reliability increased between it's last two versions i.e. 2.7 and 2.6 as seen in fig 5.6d and quantified in table 5.3.2. The table 5.3.2 shows number of weeks up to which each version shows at least 90% reliability and reliability growth that was calculated by taking this 90% reliability as standard value between the versions. The reliability growth between the version is calculated by subtracting the last week with 90% reliability for old version from the last week with 90% reliability for current version and whole difference is then divided by the total range taken into account. For instance the growth for CFEngine v3.2 was calculated by [(31-20)/115] *100%, where 31 is the number of weeks for which the current version of product offered 90% reliability, similarly 20 is also the numbers of weeks for which the old version of product offered 90% reliability and 115 being total weeks taken in account for bug tracker of CFEngine. The total weeks for Chef and Puppet were 150 weeks and 200 weeks respectively.



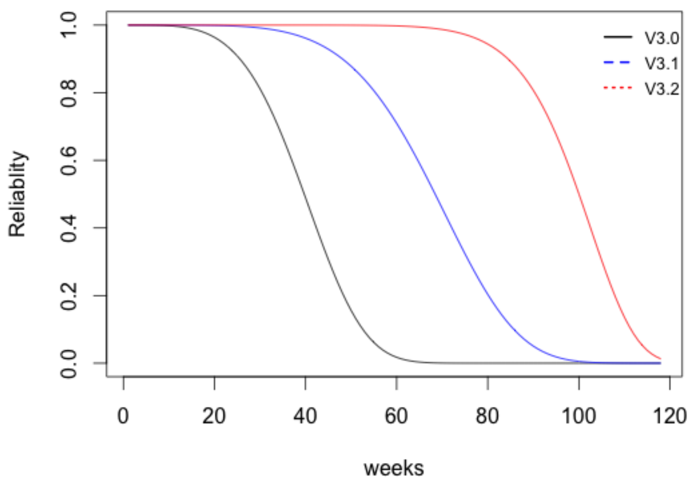
(a) version 3.0.x



(b) version 3.1.x

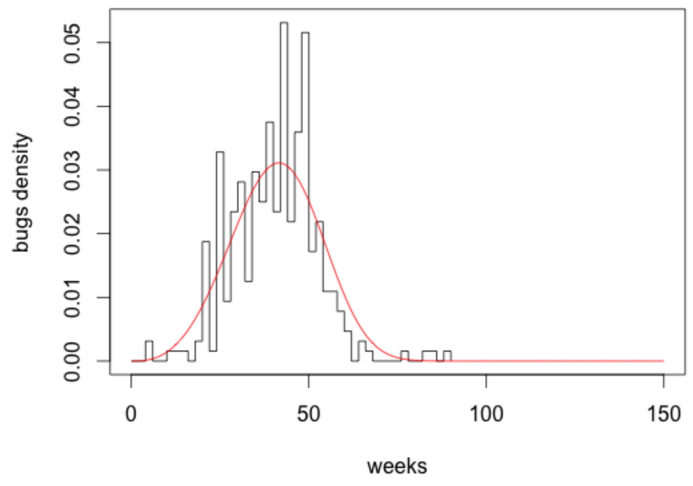


(c) version 3.2.x

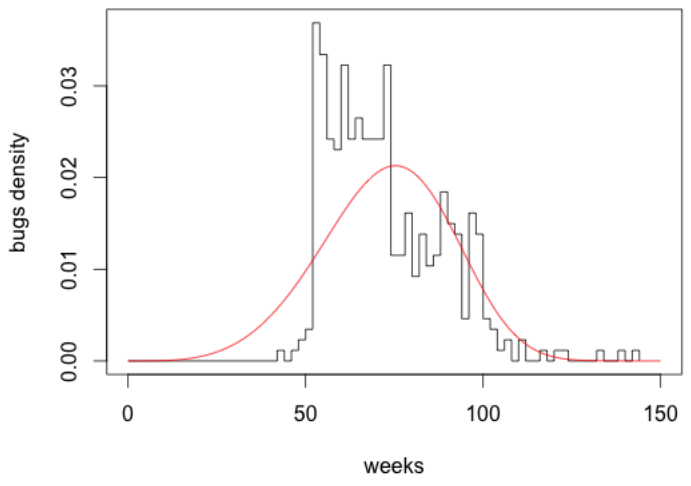


(d) Reliability growth

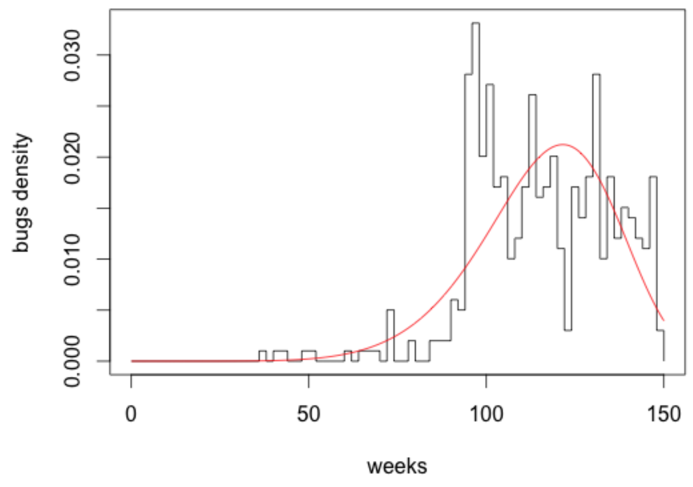
Figure 5.4: Weibull curve fitting for CFEngine versions (a), (b) and (c) and it's reliability growth (d)



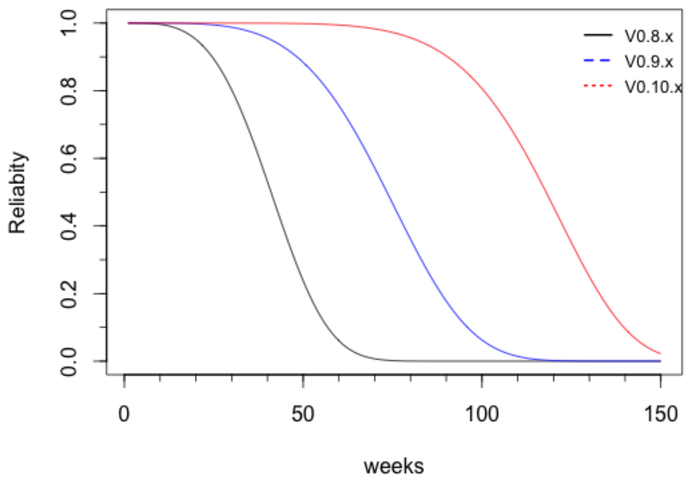
(a) version 0.8.x



(b) version 0.9.x

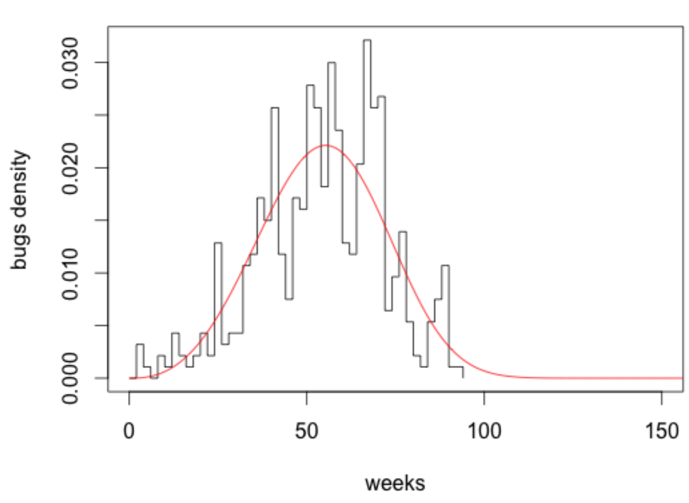


(c) version 0.10.x

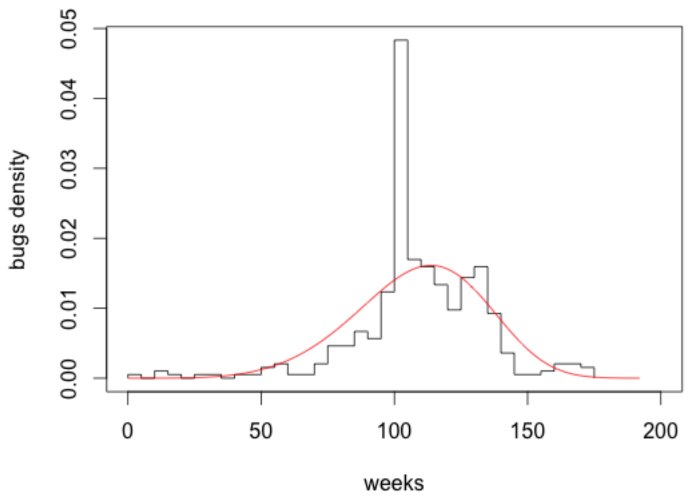


(d) Reliability growth

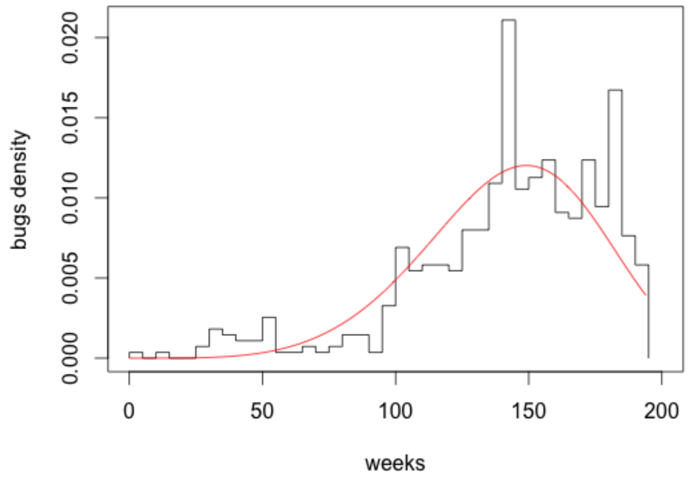
Figure 5.5: Weibull curve fitting for Chef versions (a), (b) and (c) and it's reliability growth (d)



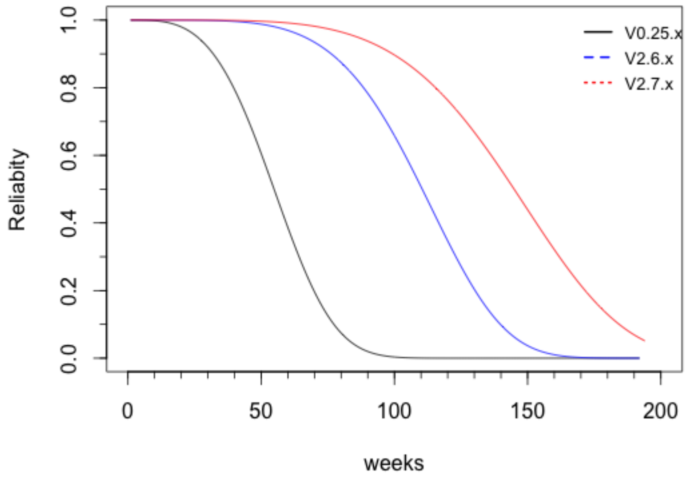
(a) version 0.25.x



(b) version 2.6.x



(c) version 2.7.x



(d) Reliability growth

Figure 5.6: Weibull curve fitting for Puppet versions (a), (b) and (c) and it's reliability growth (d)

Product Versions	Last week with at least 90% reliability	growth %
CFEngine v3.0	25	-
CFEngine v3.1	48	20%
CFEngine v3.2	84	31 %
Chef v0.8	24	-
Chef v0.9.x	48	16 %
Chef v0.10.x	90	28 %
Puppet v0.25.x	32	-
Puppet v2.6.x	76	22%
Puppet v2.7.x	99	11 %

This less increase in reliability between two version of puppet can be well explained by observing the fig4.7 as the bugs for these two version continue to arrive for a long span of time , and the time span for those two version are also almost overlapped. While for Chef and CFEngine large portion of bug reports are posted in short interval of time and as time goes on and when new version comes out the chances get even slimmer. Both CFEngine and Chef demonstrate identical and strong reliability increase in it's consecutive version but , the last version i.e. 3.3.x of CFEngine shows more growth in reliability compared to the lasy version of Chef i.e. 0.10.x which is shown in table 5.3.2. It's also well represented by the figures 5.4c and 5.5c, as the bugs for the chef's last version are spread over a large span of time in time range and the bug frequency in each week is not that high as well, so it has also a better fit index 50.80 compared to 69.80 of CFEngine. For CFEngine ,the bugs frequency are very high each week and spread over a small interval of time which resulted a small scale parameter value and thus resulted high reliability.

5.4 User experience

The raw data obtained after the test is to be processed further i.e. transformation to get a suitable data in order for it to be analyzed. Various mathematical model comes handy in analyzing these set of data , but since test sample is small caution need to be taken on application as the application of wrong theoretical model will give wrong projection of actual use case associated with these products.

Completion rate with usability problems

The task data obtained from result section is broken down into appropriate format i.e. if there was task times for 3 users then 3 out of 5 completed the test. Normally with small sample sizes it would be wrong to say that a product is 100% usable when all users completes the task and 0% usable when all users fail the test. This kind of result is often obtained in test with small sample sizes and reporting 100% and 0% doesn't portray true picture, instead it is helpful to be analyzed as how likely the true population parameter shows the value as extreme as this. Hence a point estimation method is required in order to make a estimate of the unknown population parameter value. Therefore Laplace method [48] is used as best point estimation which is suitable for small samples.

When ever 5 users out of 5 successfully completes the task, normally it is termed as 100% instead Laplace Law of Succession adds one to the numerator and two to the denominator ((x+1)/(n+2)) thus 6/7 is obtained which is 85% completion rate. This is far less than 100% but near to it. The interesting aspect of this method is that results come more closer to hundred percent when sample size increases. The height of bars in chart shown in figure 5.7 shows the point estimated by using Laplace method on the individual task data.

Though the point estimation method provides the value that will give an estimation of population parameter, the chances that the estimated value is same as the unknown population's true completion rate is extremely unlikely. Hence confidence interval also need to be computed so that it will give a reasonable boundary for a true population completion rate. For example if 1 out of 5 users completes the task , then the 95% confidence interval is 2%-64 %. Though the interval is large which is again due to sample size the important information that is obtained is that it is highly unlikely the task has a completion rate of 70%. There fore confidence interval is highly informative and is extremely necessary incase of small sample size leading to the extreme results like 0% and 100% as completion rate for the tasks conducted.

For computing confidence interval , adjusted Wald interval is utilized as it provides the best coverage for specified interval when sample size is less than 150. In other words if 95% confidence interval is desired then it will yield an interval that will contain observed proportion on average about 95 % of times [49]. To obtain an interval using this method, small adjustment is to be made in the result. For example for obtaining 95% confidence interval on 3 out of 5 i.e. 3/5 users completing the test successfully , add half of the squared z-value to numerator and squared z value to the denominator such that $\frac{(3+1.96^2/2)}{5+1.96^2}$ resulting approximately 5/9. Now the confidence interval is calculated using wald's interval on these computed values. The bar chart 5.7 shows the confidence interval obtained in this way on every bars. These confidence interval gives important information and idea about where the true completion rate for these tasks associated with the products might fall. Large set of usability studies found that average task completion is 78% which can be used as reference point to compare the results.

From the figure 5.7 shows CFEngine have the highest completion rates for all the three task but only completion rate for task 1 using CFEngine is above average. Due to three users having moderate knowledge about CFEngine, the completion rate results might have favored CFEngine. Since there is significant amount of overlap in confidence interval in completion rate on same task using these products, a conclusive result cannot be drawn rather a relative projection

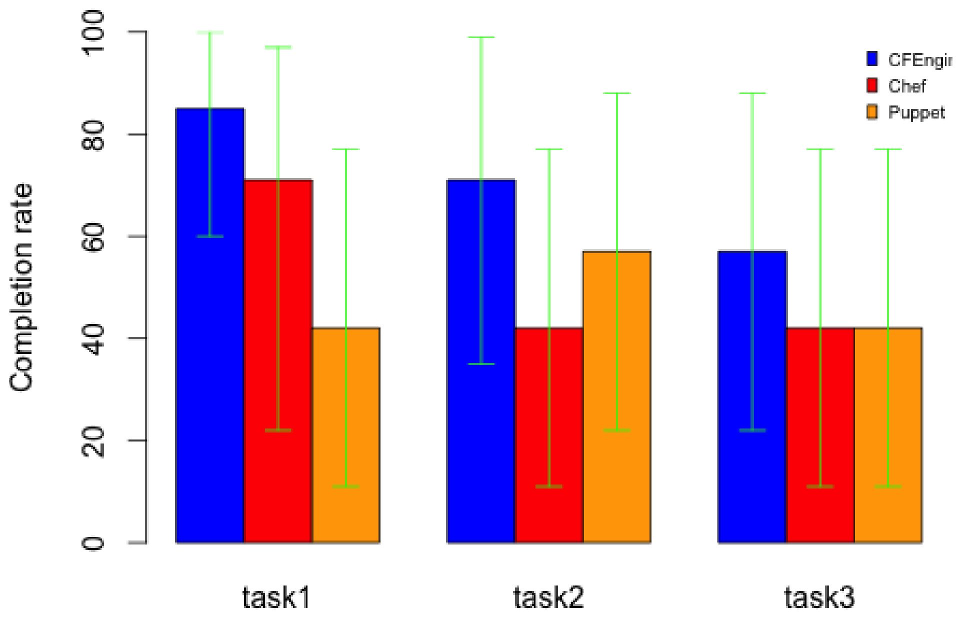


Figure 5.7: Completion rate for CFEngine, Chef and Puppet

can be made looking at the upper and lower limits of interval. Therefore, the upper limit of 95% confidence interval for all the task shows that all these products having a value greater or equal to the average task completion. This means that if examined in true population, there is possibility that all of the products can achieve average completion rate. Additionally the low value of lower limit of confidence interval shows that puppet really has less number of completion rate for task 1, while chef has less completion rate for task 2 and both chef and puppet have similar completion rate for task 3.

Considering lower limit of 95% confidence interval along with the best estimate for CFEngine, shows CFEngine having satisfactory completion rate for task 1 i.e. installation of CFEngine has completion rate that can go as low as 60% which is close to average completion rate and task completion can go up to 100%. Chef also have good completion rate regarding task one but if lower confidence interval is observed this task can have really low completion rate as low as 22% which doesn't seem satisfactory. Among the three puppet has worst completion rate for task1 which can be well explained by the table 4.7. Since majority of the test user struggled setting up Puppet server to connect with client due to certification issue triggered by DNS they were unable to complete the task. Puppet's upper limit of 95% CI shows 77% indicating it is highly improbable to get it's completion rate above the average rate. If puppet is to get this rate up in the levels of CFEngine and Chef it needs to work in the certificates issues faced by majority of users while setting up puppet.

For task 2 users had problem regarding understanding the documentation in Chef and CFEngine. The high task completion rate for CFEngine is largely due to majority of users previously having knowledge about CFEngine. But a new user getting the task completed with CFEngine signifies this task that could be conducted by looking into examples with CFEngine. Only users having problems understanding the architecture couldn't complete the problem. Thus is also well explained by bar chart with it's confidence interval between 97-35, hence it is possible to get a good task completion rate of 97% for this task using CFEngine, at the same time it can also get worst for inexperienced users. For puppet due to lot of users facing certificate issues in 1st task seems to have impact. 2 out of 5 users gave up on this task using Puppet. For chef despite the users being able to make chef working, too many terminologies involved in it seems to prevent the users from getting the task completed and finally gave up on the task. User stated problems like time consuming and confusing process seems to play the part for chef to obtain the lowest completion rate compared to others. For this task it is Chef for whom the completion rate is extremely unlikely to surpass average completion rate.

The same story being continued with puppet and chef, users giving up the task 3 due to problem in understanding the technology and lack of helpful documentation to complete task. Is also shows users giving up task 3 using CFEngine due to the same problem as with chef and puppet. Still the completion rate is higher and there is possibility for the completion rate to exceed average completion rate for CFEngine while for puppet and chef they have same completion rate and confidence interval. This task required users to be a moderate users with deep knowledge about task, and no helping material found in internet so it was expected to have low completion rate for all the products.

This shows that all the products definitely need some learning in order to go forward with the technology, while CFEngine having less components less things to configure , majority of users quickly get the task done but different users faced different problem with CFEngine, while for chef due to large number of components and many configuration some users failed the task and Finally for puppet the most of the users face the same problem and failed to over come the problem, finally end up giving up consecutive task.

5.4.1 Task time and Perceived Easiness

To get an overview of task easiness, users were asked to answer single ease question after every task. The scale points obtained from 5 users is shown in the results section 4.6. These data can to be transformed into percentage scale and calculate a mean out of them for each particular task to get a measure of users perceived easiness and satisfaction regarding the task. For example if a user gives a score of 5 , the person had build all his exception and experience abut the task in this score. Thus obtained score need to be averaged from all the users to get a single value describing the task easiness. Though this single value is easy to explain , it might not always reflect the true value , hence Confidence interval can be calculated in order to make a reasonable range that describes the task easiness and statically contains the true value for task easiness for product. Since small sample was used i.e. Less than 30, For this case estimation of confidence interval to contain real value for a unknown population will be calculated utilizing the T-distribution. The table shows the mean task easiness and confidence interval computed for each task for three products.

Had the sample size been larger we could have standardized the raw scores into z score and corresponding percentiles could have been calculated. That would have made the result much easier to understand but since the sample size was far less than 30 we could not utilize normal statistical method to get a conclusive results about the perceived easiness

Tasks	CFEngine	Chef	Puppet
Task 1	5.2*(3.58,6.81)*	2.2*(-0.02 ,4.42)*	3*(0.36,5.63)*
Task 2	5*(1.95 ,8.04)*	3.4*(-0.17,6.97)*	4*(0.59,7.40)*
Task 3	2.4*(0.73,4.06)*	2.4*(-0.83 ,5.63)*	2.6*(-0.25, 5.45)*

Unfortunately the calculated of mean and confidence interval also isn't that much helpful in interpreting the scores collected by from the participants. The basic problem with mean was scores had a high variance, and the mean could not summarize the scores. Additionally since the confidence interval sizes were also very large ranging from lower limit of the score to even in -ve territory to upper limit, it is difficult to interpret the results. The large size of confidence interval is due to small sample size and high variance of scores given by the users. Had the sample size been larger, even though the scores

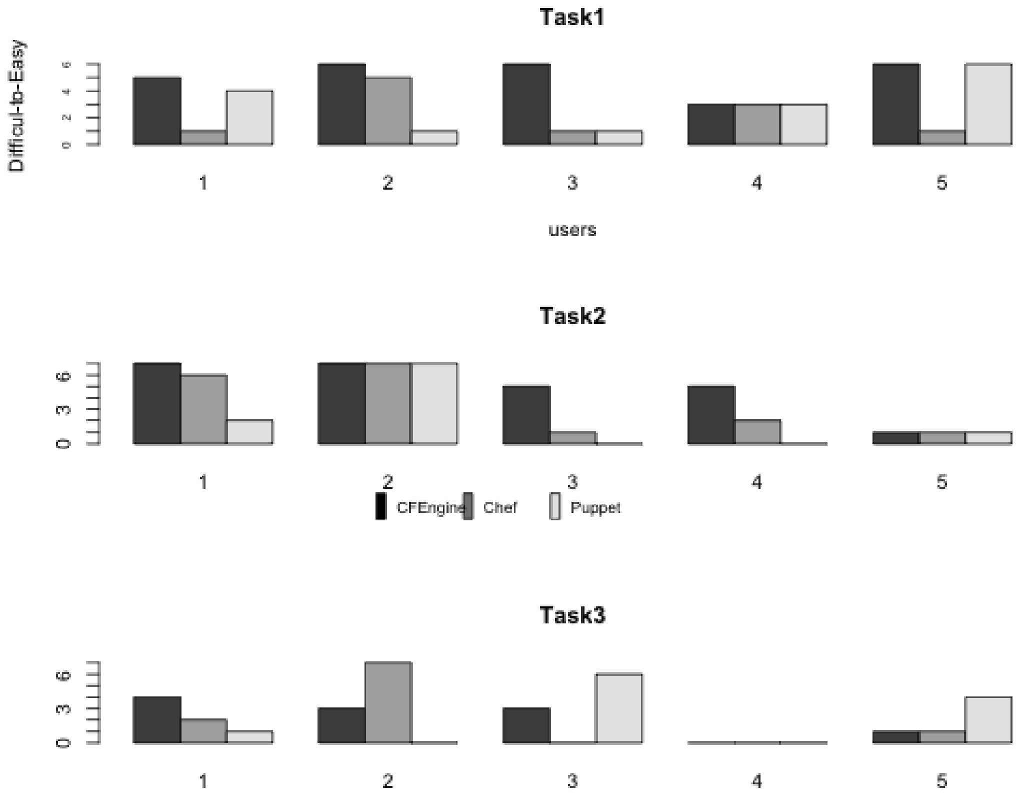


Figure 5.8: Scores for easiness level for each products

have high variance, frequency of each scores could have been examined to see the easiness level of each task.

So to present data in perceivable way , figure 5.8 shows bar chart that allows to see the comparison of task easiness between these products. The figure is even easier to comprehend when the statistics about task time is considered and try to understand the why was the score given by the user. More importantly it could lead to interpretation of what scores level represents and which scores levels are used in relation to the task time. The table 5.4.1 shows the correlation between task time and scores given by each user for the the task they performed. It also shows the geometric average of the task times required by each users for performing the task using CFEngine, Chef and Puppet.

Tasks	Correlation(task time and scores)	CFEngine	Chef	Puppet
Task1	-0.373	17.66	37.39	32.76
Task 2	-0.178	14.84	16.28	22.83
Task 3	-0.72	37.45	16.94	16.73

The correlation between the scores and the task times shows mixed results for the tasks. While users seemed to have strong opinion about the task3 and task1, their perceived satisfaction seemed to vary in task2. It is obvious to have a negative correlation between the task time and task score as if the task is completed quickly it is easier and hence will obtain a high score. But in task 2 , even with an low amount of average task time the task scores is not able to show users perceived satisfaction in correlation. As seen in figure 5.8, almost every users have assigned a scores, some users get the task completed in small amount of time leading to high amount of perceived satisfaction , while others are spending a large chunk of time to get the task completed. But the users taking large amount of time doesn't seem to be turned off by product as they are getting the task implemented, hence they also assigned a satisfactory score to the problem which seems to deter the correlation between task time and task score. In conclusion users point of view about the task was, it is do able with a satisfactory level of satisfaction in all products.

For task1 it shows an moderate negative correlation , which points out that users scores reflects their experience. Since this was just installation , users expect this task to be done by simply following the documentation. But when users take a lot of time and encounter problems while doing simple installation , it had impact on users perceived satisfaction. As seen in the figure 5.8 , for task 1 every users have reported their scores and CFEngine seems to do well in obtaining high scores. This is supported by the average task time result as well. The task time needed for CFEngine is almost half the time need for Chef and Puppet. So with this time about 17 mins , users seems to assign a level 5-6 scores in easiness scale. Therefore it can be understood that if a user is able to make a server and client connected in about 15 minutes then it results in high level of satisfaction. While for Chef the scores is around 3-5 and the task time around 38 minutes. Therefore in this case, the scores 3-5 represent a satisfactory level. And for Puppet the score is around 1 and only two users being able to complete the task, though the task time was just 32mins. Therefore for this task even though the task is do able in under a satisfactory level, lot of users fails in task propelling puppet is into unsatisfactory level.

Task3 shows a strong correlation between task time and task scores. Thus the scores for it reflect true picture about user experience compared to prior two tasks. The average task time for CFEngine is 38 mins and it's task scores is at level 3 and below. Hence user has perceived this task with high difficulty level compared to prior task performed by CFEngine. Also some users failing to report the scores in figure 5.8 signify that they gave up the task or did not tried it at all. This is also the strong indication of low satisfaction. Even with the users with CFEngine experience rating the task in level 3 and requiring a lot of time to complete the task is another symptom of low level of perceived satisfaction. For Chef, the average task time is low and a lot of users failing to complete the task coupled with task score ranging 1-7 signify that chef offers a high satisfaction if users knows the product and can get the task completed. But Majority of users giving low scores signifies that is hard to get the task completed with chef reducing satisfaction level. Similar is the case with puppet, with majority of users failing to get the task implemented and some users giving high scores coupled with small average task time hints the same thing. Puppet offers high amount of task satisfaction if users gets the task done or know it.

5.4.2 Over all product usability

The calculation of raw scores into SUS scores had yielded a single number representing a composite measure of overall usability as perceived by the users. The individual scores for the product by the users is shown on table 4.8 which needs to be averaged together in order to get a average SUS score for the product. Since sample was small and the SUS scores are obtained have a very high variability i.e. from low to high , if a mean is calculated then it will get skewed towards higher value. Hence as pointed out by background study, for small sample sets geometric mean gives a most reasonable middle point for a set of data. Should the sample size grow larger and data obtained is more uniform the obtained mean,median and a geometric mean will be similar. Therefore it is much suitable for getting single SUS score for a product from a set of SUS scores obtained from users. Thus calculated averaged SUS scores for product is shown in below

Products	**SUS scores**
CFEngine	57.77
Chef	19.41
Puppet	42.74

Now utilizing the figure 2.6 corresponding percentile and grades for the product usability could be obtained. CFEngine have percentile score of about 23 which has a grade score of D. Thus CFEngine only has a perceived usability that is higher than 23% of the products. Though the SUS score 57 is near to average usability score 68 having 51 percentile , it is far less in it's percentile ranks on it's usability. The grade D is explainable for CFEngine, as when it comes to do a complex task it might involved a large amount of resource consumption i.e. (time and mental effort) to get the task done which too is demonstrated by task 3. Both Puppet and Chef obtains a grade F in terms of their perceived usability with Chef having the least SUS scores. The low score obtained for Chef is might be because of the various technologies and terminologies used in chef, and chef having multiple components to be configured to get working. Majority of users were not able to complete all the task using chef which is also a add to this low usability score. Puppet's score is close to a CFEngine but users still fill less satisfied with it , it might be because of users facing problem early on task 1 and majority of them not being able to solve the issue despite their best effort, after they get around the problems the tasks became are easier.

If conversion rate is calculated ,it is observed that 3 out of 5 users have carried out all task using CFEngine and 2 out of 5 users have completed all task using Chef , and 1 out of 5 users have completed all task using Puppet. Even with high amount of conversion rate of Chef compared to Puppet , it has low SUS score , signifies that it is much more complex system to work with compared to CFEngine and Puppet. Additionally even though the users in test have some experience with CFEngine it did not seem to do to well i.e. it is not meeting the average usability hints that it is also difficult system to work with. For puppet it's usability is unstable , people that can make it works are happier but others will have a frustrating time working with product. In our test case , majority users seem to have a frustrating time with puppet which ended up in giving up task with puppet.

Chapter 6

Conclusion and Future work

The investigation of unexplored aspects of configuration management systems like CFEngine, Chef and Puppet was quite interesting and informatory. The aim of this thesis was to find out if any one of the three tools stands out when it comes to measure complex characteristics like community, usability e.t.c. Up until now the prospective users of CM tool have to rely on word-of-mouth and buzz to create an image of what they think is a better product on these aspects i.e community , usability e.t.c. But this study hopes to shed light in those aspects and ultimately answer their curiosity. The focus was given on collection of whatever data found on web regarding these product and to get as much information undergoing in-depth analysis, finally to obtain a measurable quantity that can sum up findings with the use of theoretical models.

Community has often been linked as measureless quantity and impossible to apprehend. People associate community in terms of numbers of users involved with product's development, usage, discussions and reviews. There are different field of study like support ,development , popularity , usage, collaboration etc. that falls into community which makes it difficult to apprehend. Explaining community as a single entity is almost impossible. Thus different aspects within the community needs to be explored separately. In this paper almost every aspect of the community except development is explored to get a conclusive result. Popularity was investigated by usage of google trends which showed that currently all of the three (CFEngine, Chef and Puppet) are equally popular despite these products having different maturity level however Chef shows high rate of popularity growth in short interval of time.

The firm understanding of amount of resources (discussion and articles) that can be found for a product is both helpful for learning as well as problem discovery and problem solving. Hence the popular information social discussion platform named "haker news" was analyzed to get overview of resources that might be available to users. The results showed Puppet has large the numbers of articles and people talking about the product, followed by Chef and CFEngine. One thing to notice was despite Chef being much younger, there were large number of discussions happening around it close to that of Puppet, while the results also showed that any search result obtained when looking for ″CFEngine″ word doesn′t contain any inappropriate articles or discussions.

The categorization of community members was carried out by inspection of mailing list messages. Mailing list captures the interaction of community members that makes it an ideal source to investigate the community structure and understand the support provided by it′s members. It was found that nearly every product have a similar proportion of individuals categorized as seekers, providers and both. This signified that they all have similar community structure despite the difference in community size. The activities like question posting , answering to question etc. of individuals under these group were also similar. Another interesting finding was for all the products, the members in mailing list were not equally participative in answering queries. Therefore, support activity provided by the community exhibited exponential behavior where large number of individuals were answering small proportion of questions and small proportion of individuals responding to large number of threads which was quantified by Lorenz curve of inequality. Thus calculated inequality metric called gini index was not significantly different for these product. This demonstrates that support obtained from community doesn′t depend on community size, at least not in mailing list, but community size has strong influence on resources available for the product.

Failure to execute an assigned task decreases reliability of configuration tool. Bugs are the primary source of failure and bug reports provide better perception of failure distribution that helps to figure out the reliability of a software. In this thesis, data from bug repositories for CFEngine , Chef and Puppet was collected and Weibull distribution was fitted on observed weekly bug frequencies. It was observed that even though the goodness of fit was not satisfactory, the model was able to explain the arrival pattern of bug reports. There was significant overlap between the bug reports of two different versions for puppet , while CFEngine and Chef showed least overlap between these versions. The later versions for all the observed products showed increase in reliability. Latest version of CFEngine showed high amount of reliability gain followed by Chef and then Puppet. Over all the amount of reliability increase between subsequent versions on all these product did not show much difference.

User experience with software defines the usability. Hence in this study usability was investigated by capturing objective and subjective aspect of the user experience with software. An usability test was conducted with small sample of users. Statistics about task time , completion rate and users opinion about task and product as a whole was gathered. Due to small sample size the result could not give a conclusive answer on product offering best usability, however the experiment was successful in identifying the common problems associated with each product and over all user′s perception about these products. Users faced same problem with puppet, but with Chef and CFEngine different users faced different problems. It was also discovered that not a single product was able to provide consistent and stable user experience across all the users through out all the task. There was large variation of users opinion on each task level but on product level there was no significant difference in users opinion. Each product has some learning curve and posses difficulty in one form or another. In summary all of the observed product had perceived satisfaction below average. When it comes to usability none of them is exceptionally good compared to each other.

Because of the time invested in collection and analysis of information on wide range of aspects, some of the sources and work that could provide interesting insights into each aspect could not be covered. Therefore a possible future research would be to determine the resources available and understand the support as well as product popularity , commonly known question answer platform like "stack overflow" and popular public repository site like "GitHub" could be investigated. Another prospective future work would be to consider developer mailing list also as a bug source and conduct the combined study of feature growth with reliability growth.

The usability test could be conducted with large number of participants, most importantly more than 30 to investigate further in the frontiers of usability. Conducting the test with larger sample increases the precision of estimates resulting narrow confidence interval from which conclusive results can be extracted related to task completion rate and task satisfaction, which possibly can help to identify the product that offers best usability.

This kind of research can lead towards well established methods for investigating community, reliability and usability of a software, layout the platforms for making comparison of these attributes within different softwares and also to be used across different software disciplines.

Bibliography

[1] Cory Lueninghoener. Getting started with configuration management. *USENIX*, 2011.

[2] Aleksey Tsalolikhin. Configuration management summit. Technical report, Vertical Sysadmin, 2010.

[3] Sebastian Carlier Niek Timmers. Automating configuration cfengine vs puppet. 2010.

[4] Jarle Bjorgeengen. Puppet andcfengine compared: time and resource consumption. 2010.

[5] W . Joosen T . Delaet and B . Vanbrabant. A survey of system conguration tools. 2010.

[6] Roberto Galoppini. How to evaluate open source software / free software (oss/fs) programs. Technical report, 2010.

[7] David A. Wheeler. How to evaluate open source software / free software (oss/fs) programs. Technical report, 2011.

[8] Open source software assessment methodologies.

[9] Method for qualification and selection of open source software (qsos) version 1.6. Technical report, Atos Origin, 2006.

[10] Business readiness rating for open source. Technical report, brr.org, 2005.

[11] Qualoss,. Technical report, 2008.

[12] Simon Alexandre Jean-Christophe Deprez. Comparing assessment methodologies for free/opensource software: Openbrr and qsos1.

[13] Stephen O'Grady. Community metrics: Comparing chef and puppet. Technical report, RedMonk, 2012.

[14] Jarle Bjorgeengen. Feature comparison puppet, redhat satellite server and cfengine. 2009.

[15] Akita. Chatting with adam jacob. Technical report, Codeminer 42, 2010.

[16] Azad Azadmanesh Cobra Rahmani, Harvey Siy. An experimental analysis of open source software reliability*.

[17] Alesky. Getting started with cfengine. 2011.

[18] *CFEngine Refrence Manual*.

[19] J. Crowston, K. & Howison. The social structure of free and open source software development,. , 2005.

[20] Eric von Hippel Karim R. Lakhani. How open source software works: free user-to-user assistance. Technical report, MIT Sloan School of Management,, 2002.

[21] J.D. Musa and K. Okumoto. A logarithmic poisson execution time model for software reliability measurement. In .

[22] Azad Azadmanesh Cobra Rahmani, Harvey Siy. An experimental analysis of open source software reliability. Technical report, University of Nebraska-Omaha, 2009.

[23] *Software reliability research*. Stattitical Computer Performance Evaluation, Academic Press, Inc., New York, 1972.

[24] B. Littlewood and J.L. Verrall. A bayesian reliability model with a stochastically monotone failure rate. *IEEE Transactions on Reliability*, 1974.

[25] A.L. Goel and K. Okumoto. A time-dependent error-detection rate model for software reliability and other performance measure. *IEEE Transactions on Reliability*, 1979.

[26] Dennis J. Wilkins. The bathtub curve and product failure behavior.

[27] Jiantao Pan. Software reliability.

[28] Joseph DAVIS Ying ZHOU. Open source software reliability model: an empirical approach.

[29] Nigel Bevan. Measuring usability as quality of use. Technical report, NPL Usability Services, National Physical Laboratory, 1995.

[30] JEFF SAURO. Measuring usability. Technical report, Measuring Usability LLC, 2004.

[31] JEFF SAURO. Deriving a problem discovery sample size. Technical report, RedMonk, 2012.

[32] Joseph S. Dumas Jeff Sauro. Comparison of three one-question, post-task usability questionnaires. Technical report, Oracle Corporation,, .

[33] John Brooke. Sus - a quick and dirty usability scale. Technical report, Redhatch Consulting Ltd., 1986.

[34] J.R. Lewis and J. Sauro. The factor structure of the system usability scale. In .

[35] Kortum Philip T. Bangor, Aaron and James T. Miller. An empirical evaluation of the system usability scale. *nternational Journal of Human-Computer Interaction*, 2008.

[36] Bryan Berry. Puppet vs. chef, fight! 2011.

[37] Klint finley. Puppet vs. chef: Which is more popular. 2011.

[38] Andrs. Analysis on free software communities (i): a quantitative study on grass, gvsig and qgis. 2011.

[39] Robert A. Muenchen. The popularity of data analysis software. 2011.

[40] Mario Garzia Ben Errez Pankaj Jalote, Brendan Murphy. Measuring reliability of software products.

[41] *Practical Reliability Analysis*. Prentice Hall, 2004.

[42] Sampling distributions.

[43] Hypothesis test for a proportion.

[44] M. E. J. Newman. Power laws, pareto distributions and zipfs law. Technical report, Department of Physics and Center for the Study of Complex Systems, University of Michigan, Ann Arbor, MI 48109. U.S.A., 2006.

[45] Avinash Narula. What is 80/20 rule?

[46] M. O. Lorenz. Methods of measuring the concentration of wealth. *Publications of the American Statistical Association*, .

[47] Vito Ricci. Fitting distributions with r. 2005.

[48] James R. Lewis Jeff Sauro. When 100estimates of completion rates. *Journal of usability study*, 2006.

[49] James R. Lewis Jeff Sauro. Estimating completion rates from small samples using binomial confidence intervals: Comparisons and recommendations. In *PROCEEDINGS of the HUMAN FACTORS AND ERGONOMICS SOCIETY 49th ANNUAL MEETING2005*.

Appendix A

HTML Parser and Crawler

A.1 For Puppet's Mailing List

```
#!/usr/bin/perl

# Needed packages
use Getopt::Std;
use Date::Parse;
use Scrappy;
use DBI;
use strict "vars";

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

# DataBase Connection
my $db="puppetMLS";
my $host="localhost";
my $user="username";
my $password="password";  # the root password

# handle flags and arguments
# Example : c = "-c", c: = "-c argument"
my $opt_string = 'vdh';
getopts("$opt_string",\my %opt) or usage()and exit 1;

# Print help message in -h is invoked
if($opt{'h'}){
    usage();
    exit 0;
}

# Handle other user input
```

```
$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
verbose("Verbose is enabled\n");
debug("Debug is enabled\n");


###### Main  script content
#
# my $file=$opt{'f:'};
  my $base_url="http://markmail.org";
  my $url="http://markmail.org/browse/com.googlegroups.puppet-users";
  my @visitedUrls;
# Open Connection to databse


  my $scrapper = Scrappy->new;

  if($scrapper->get($url)->page_loaded){
     my $yearmth = $scrapper->select('table table tr a')->data;
     my $values = $scrapper->select('table table tr td:nth-child(2)')->data;

     ##################################
     # Saving  the message activity  in file
     #################################
     open (OUT , ">puppetdata.csv") or die "Cannot open the file for writing";
     my $i=0;
      foreach my $value (@{$values}){
  my $date = format_date($yearmth->[$i]->{text});
  print OUT $date.",".$value->{text} ."\n";
          # print $scrapper->dumper($date);
          $i++;
      }
      close(OUT);

      ################################
      # Load the previously traversed links
      ################################
      my $dbh = DBI->connect ("DBI:mysql:database=$db:host=$host",$user,$password) or
      my $sth = $dbh->prepare("select url from messages");
      $sth->execute();
      while (my @row = $sth->fetchrow_array()){
      push(@visitedUrls,@row);
  }
      $dbh->disconnect or warn "Disconnection error: $DBI::errstr\n";

      ##################################
      # Get the links and start visiting
      ##################################
      debug("Visting the links in first page");
      my $ymScrapper=Scrappy->new;
      my $linkCount=0;
      foreach my $link(@{$yearmth}){
 # last if($linkCount>0);
  my $suburl= $link->{href};
     visitArchiveYearMonth($suburl);
     $linkCount++;
      }
  }
#
#########
sub visitArchiveYearMonth{
  my $ymScrapper = Scrappy->new();
    my @matched = grep(/^$_[0]$/,@visitedUrls);
    my $matchedno = @matched;
   # print $matchedno."\n";
   if($matchedno == 0){

       push(@visitedUrls,$_[0]);
       if($ymScrapper->get($_[0])->page_loaded){
       debug("Loaded Year Month :". $_[0]);
       my $threads=$ymScrapper->select('#browse table tr a')->data;

       my $count=1;
       foreach my $thread(@{$threads}){
         collectThreadMessages($thread->{href},$thread->{text});
}
       }


       my @urlparts =split('/',$_[0]);
       my $size=@urlparts;
       if($size < 7){
   my $pages=$ymScrapper->select('#browse h4 a')->data;
   foreach my $page(@{$pages}){
        # print $page->{text}."\n";
        visitArchiveYearMonth($page->{href});
   }
       }
    }
 }


sub collectThreadMessages{
    my $threadCrawler = Scrappy->new();
    my @matched = grep(/^$_[0]$/,@visitedUrls);
    my $matchedno = @matched;

    if($matchedno == 0){
    if($threadCrawler->get($_[0])->page_loaded){
       debug("Loaded thread :".$_[0]);
       my $count=0;
       my $messages=$threadCrawler->select('#thread table tr a')->data;
       foreach my $message(@{$messages}){
  # last if($count > 1);
   CollectMessageDetails($message->{href},$_[1]);
   $count++;
       }
    }
  }
}

sub CollectMessageDetails{
    my $messageScrapper = Scrappy->new();
    my $dbh = DBI->connect ("DBI:mysql:database=$db:host=$host",$user,$password) or di
    # $dbh->trace(2);
    my @matched = grep(/^$_[0]$/,@visitedUrls);
    my $matchedno = @matched;
    print $matchedno."\n";

    if($matchedno == 0){
    push(@visitedUrls,$_[0]);
    my $sth = $dbh->prepare("INSERT INTO messages(subject,sender,send_date,message,thr
    if($messageScrapper->get($_[0])->page_loaded){
       debug("Loaded message: ".$_[0]."\n");
       my $subject = $messageScrapper->select('#headers tr')->focus(0)->select('td:nth
       my $from = $messageScrapper->select('#headers tr')->focus(1)->select('td:nth-ch
       my $date = $messageScrapper->select('#headers tr')->focus(2)->select('td:nth-ch
       my $lines = $messageScrapper->select('#body div.pws > p')->data;
       my $message="";
       my $formatted_date = format_date_for_mysql($date);
       foreach my $line(@{$lines}){
    $message.=$line->{text};
       }
       $sth->execute($subject,$from, $formatted_date, $message,$_[1], $_[0]);
    }

   }
   $dbh->disconnect or warn "Disconnection error: $DBI::errstr\n";
}
```

```
sub format_date{
   my %mon2num = qw(
  jan 1  feb 2  mar 3  apr 4  may 5  jun 6
  jul 7  aug 8  sep 9  oct 10 nov 11 dec 12
);
    my @datesupplied = split(' ',$_[0]);
    my $month = $mon2num{lc substr($datesupplied[1],0,3)};

    return $datesupplied[0].'/'.$month;
    }

sub format_date_for_mysql{
   my @date=strptime($_[0]);
   my $year=$date[5]+1900;
   my $month=$date[4]+1;
   my $day=$date[3];
   my $hh=$date[2];
   my $mm=$date[1];
   my $ss=$date[0];
    return "$year-$month-$day $hh:$mm:$ss";
}

sub usage{
    # prints the correct use of this script
      print "Usage:\n";
      print "-h Usage \n";
      print "-v Verbose\n";
      print "-d Debug\n";

      print "./script [-d][-v][-h]\n";
}

sub verbose{
    print "VERBOSE:".$_[0]."\n" if $VERBOSE;
}

sub debug{
    print "DEBUG:".$_[0]."\n" if $DEBUG;
}
```

A.2 For Chef's Mailing List

```
#!/usr/bin/perl

# Needed packages
use Getopt::Std;
use Date::Parse;
use Try::Tiny;
use Scrappy;
use DBI;
use POSIX;
use strict "vars";

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

# DataBase Connection
my $db="chefMLS";
my $host="username";
my $user="password";
my $password="letmein";  # the root password

# handle flags and arguments
# Example : c = "-c", c: = "-c argument"
my $opt_string = 'vdh';
getopts("$opt_string",\my %opt) or usage()and exit 1;

# Print help message in -h is invoked
if($opt{'h'}){
    usage();
    exit 0;
}

# Handle other user input
$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
verbose("Verbose is enabled\n");
debug("Debug is enabled\n");


###### Main  script content
#
# my $file=$opt{'f:'};
  my $base_url="http://markmail.org";
  my $url="http://lists.opscode.com/sympa/arc/chef";
  my @visitedUrls;
  my @yearMths;
  my %YMmessages;
# Open Connection to databse


  my $scrapper = Scrappy->new;
  if($scrapper->get($url)->page_loaded){
     my $yearmth = $scrapper->select('#ArcCalendar a')->data;
     # my $values = $scrapper->select('table table tr td:nth-child(2)')->data;

     ##################################
     # Saving  the message activity  in file
     #################################
     open (OUT , ">chefMLS.csv") or die "Cannot open the file for writing";
     my $i=0;
      foreach my $link (@{$yearmth}){
  my @linkparts = split('/',$link->{href});
  my @messages = split(' ',$link->{title});
  my $date = $linkparts[6];
  my $value= $messages[0];
  push(@yearMths,$date);
  $YMmessages{$date}=$value;
  print OUT $date.",".$value."\n";
         # print $scrapper->dumper(@messages);
         $i++;
      }
     close(OUT);

      #################################
      # Load the previously traversed links
      #################################
      my $dbh = DBI->connect ("DBI:mysql:database=$db:host=$host",$user,$password) or
      my $sth = $dbh->prepare("select url from messages");
      $sth->execute();
      while (my @row = $sth->fetchrow_array()){
      push(@visitedUrls,@row);
  }
      $dbh->disconnect or warn "Disconnection error: $DBI::errstr\n";

      ##################################
      # Get the links and start visiting
      ##################################
      debug("Visting the links in first page");
      my $linkCount=0;
      foreach my $link(@{$yearmth}){
# last if($linkCount>1);
  my $suburl= $link->{href};
     visitArchiveYearMonth($suburl);
     $linkCount++;
      }
  }
#
```

```
#########
sub visitArchiveYearMonth{
  my $ymScrapper = Scrappy->new();
    my @matched = grep(/^$_[0]$/,@visitedUrls);
    my $matchedno = @matched;
   # print $matchedno."\n";
   if($matchedno == 0){
       push(@visitedUrls,$_[0]);
       my @linkparts = split('/',$_[0]);
       my $pages = ceil($YMmessages{$linkparts[6]}/30);
       # print $scrapper->dumper($pages);

       for(my $i=1; $i<=$pages ;$i++){
          # last if($i>2);
   my $pageurl=$_[0].'thrd'.$i.'.html';
   # print $pageurl;
   if($ymScrapper->get($pageurl)->page_loaded){
               debug("Loaded Year Month: ". $pageurl."\n");
               my $threads=$ymScrapper->select('#Paint > div.ContentBlock > ul a')->da
               my $count=1;
               foreach my $thread(@{$threads}){
 # last if($count>2);
           collectThreadMessages($thread->{href},$thread->{text});
   # print $scrapper->dumper($thread->{text});
   $count++;
}
           }
       }


    }
 }


sub collectThreadMessages{
    my $threadCrawler = Scrappy->new();
    my @matched = grep(/^$_[0]$/,@visitedUrls);
    my $matchedno = @matched;

    if($matchedno == 0){
    if($threadCrawler->get($_[0])->page_loaded){
       debug("Loaded thread :".$_[0]);
       # load all the sub messages with in the thread
       my $messages=$threadCrawler->select('#Paint > div.ContentBlock > ul a')->data;
       CollectMessageDetails($_[0],$_[1]);
       my $count=0;
       foreach my $message(@{$messages}){
```

```
 # last if($count > 1);
   CollectMessageDetails($message->{href},$_[1]);
   $count++;
       }
    }
  }
}

sub CollectMessageDetails{
    my $messageScrapper = Scrappy->new();
    my $dbh = DBI->connect ("DBI:mysql:database=$db:host=$host",$user,$password) or di
    # $dbh->trace(2);
    my @matched = grep(/^$_[0]$/,@visitedUrls);
    my $matchedno = @matched;
    print $matchedno."\n";

    if($matchedno == 0){
    push(@visitedUrls,$_[0]);
    my $sth = $dbh->prepare("INSERT INTO messages(subject,sender,send_date,message,thr
    if($messageScrapper->get($_[0])->page_loaded){
       debug("Loaded message: ".$_[0]."\n");
# collect message details
       my $body = $messageScrapper->content->decoded_content('charset');

# $body =~ s/(mailto[:][\n]\s*<script.*>(.*[\n]*)+<\/script>[\"]>)/\">/i;
        while($body =~ m/(<[^<>]*<script.*>(.*[\n]*)+<\/script>[^<>]*>)/){
    # +<script(.*[\n]*)+<\/script>[\"]>)/i){
    # print "matched ".$1."\n";
    $body =~ s/(<[^<>]*<script.*>(.*[\n]*)+<\/script>[^<>]*>)//i;
    }
       # print $scrapper->dumper($body)."\n";
while($body =~ m/(<a([^>]+)>(.+?)<\/a>)/i){
     $body =~s/(<a([^>]+)>(.+?)<\/a>)//i;
  }

       my  $parser = Scrappy::Scraper::Parser->new;
   $parser->html($body);
   # $parser->select('#Paint > div.ContentBlock > div.block li:nth-child(1)');

       my $from=$messageScrapper->select('#Paint > div.ContentBlock > div.block li:nth
# print $scrapper->dumper($parser->data)."\n";
       $from =~ s/.*<(.*?)>/$1/i;
       if($from ==""){
       $from=$messageScrapper->select('#Paint > div.ContentBlock > div.block li:nth-ch
$from =~ s/(.*[\n])+.*\(&quot;(.*?)&quot;\s+\W+\s+&quot;(\W+)&quot;\s+\W+\s+&quot;(.*?
       }
```

```
      my $subject=$messageScrapper->select('#Paint > div.ContentBlock > div.block li:n
      if($subject !~ m/Subject:\s+(.*?)/i){
 $subject=$parser->select('#Paint > div.ContentBlock > div.block li:nth-child(4)')->da
      }
      $subject =~ s/Subject:\s+(.*?)/$1/i;

      my $date = $parser->select('#Paint > div.ContentBlock > div.block li:nth-child(4
      if($date !~ m/Date:\s+(.*?)/i){
$date=$parser->select('#Paint > div.ContentBlock > div.block li:nth-child(5)')->data->
      }

      $date =~ s/Date:\s+(.*?)/$1/i;
      my $formatted_date = format_date_for_mysql($date);
      print $subject." ".$formatted_date."\n";

      my $msgbody = "";
        # $msgbody=$parser->select('#Paint > div.ContentBlock')->data->[0]->{html};
 # print $scrapper->dumper($msgbody)."\n";
 $msgbody =~ s/<hr[^<>]*\/>(.*?)<script.*/$1/i;
         $parser->html("<div id=\"dummy\">".$msgbody."</div>");
       my $msgtxt=$parser->select("#dummy")->data->[0]->{text};

         # $sth->execute($subject,$from, $formatted_date, $msgtxt,$_[1], $_[0]);
    }

   }
   $dbh->disconnect or warn "Disconnection error: $DBI::errstr\n";
}


sub format_date{
   my %mon2num = qw(
  jan 1  feb 2  mar 3  apr 4  may 5  jun 6
  jul 7  aug 8  sep 9  oct 10 nov 11 dec 12
);
    my @datesupplied = split(' ',$_[0]);
    my $month = $mon2num{lc substr($datesupplied[1],0,3)};

    return $datesupplied[0].'/'.$month;
    }

sub format_date_for_mysql{
   my @date=strptime($_[0]);
   my $year=$date[5]+1900;
   my $month=$date[4]+1;
   my $day=$date[3];
   my $hh=$date[2];
   my $mm=$date[1];
   my $ss=$date[0];
    return "$year-$month-$day $hh:$mm:$ss";
}

sub usage{
    # prints the correct use of this script
      print "Usage:\n";
      print "-h Usage \n";
      print "-v Verbose\n";
      print "-d Debug\n";

      print "./script [-d][-v][-h]\n";
}

sub verbose{
    print "VERBOSE:".$_[0]."\n" if $VERBOSE;
}

sub debug{
    print "DEBUG:".$_[0]."\n" if $DEBUG;
}
```

Appendix B

Data Miner and tokeniser

This script is originally written by Eric Lease Morgan . Minor tweaks like database connection , filter addition and selection of payload to be inspection were made make it work in our context

```
# n-grams.pl - list top 10 bi-grams from a text ordered by tscore, and
#              list top 10 tri-grams and 4-grams ordered by number of occurances

# Eric Lease Morgan <eric_morgan@infomotions.com>
# June   18, 2009 - first implementation
# June   19, 2009 - tweaked
# August 22, 2009 - added tri-grams and 4-grams


# require
use lib '../lib';
use Lingua::EN::Bigram;
use Lingua::StopWords qw( getStopWords );
use DBI;
use Data::Dumper;
use strict;

# initialize
my $stopwords = &getStopWords( 'en' );

# definition of variables
my $db="chefMLS";
my $host="localhost";
my $user="root";
my $password="letmein";  # the root password


# sanity check
# my $file = $ARGV[ 0 ];
# if ( ! $file ) {
```

```
#    print "Usage: $0 <file>\n";
#   exit;

#   }

# slurp
my $dbh   = DBI->connect ("DBI:mysql:database=$db:host=$host",
                                   $user,
                                   $password)
                               or die "Can’t connect to database: $DBI::errstr\n";
   my $text = " ";
   my $sth = $dbh->prepare("select thread from threads");
   $sth->execute();
   while (my ($thread) = $sth->fetchrow_array()){
      # push(@urls,@row
       $text = $text." ".$thread;
       # print $text;
       print Dumper($thread)."\n";
   }

  # disconnect  from database

$dbh->disconnect or warn "Disconnection error: $DBI::errstr\n";
# open F, $file or die "Can’t open input: $!\n";
# my $text = do { local $/; <F> };
# close F;

# build n-grams
my $ngrams = Lingua::EN::Bigram->new;
$ngrams->text( $text );

# get bi-gram counts
my $bigram_count = $ngrams->bigram_count;
my $tscore       = $ngrams->tscore;

# display top ten bi-grams, sans stop words and punctuation
my $index = 0;
print "Bi-grams (T-Score, count, bi-gram)\n";
foreach my $bigram ( sort { $$tscore{ $b } <=> $$tscore{ $a } } keys %$tscore ) {

    # get the tokens of the bigram
    my ( $first_token, $second_token ) = split / /, $bigram;

    # skip stopwords and punctuation
    next if ( $$stopwords{ $first_token } );
    next if ( $first_token =~ /[,.?!:;()\-]/ );
    next if ( $$stopwords{ $second_token } );
    next if ( $second_token =~ /[,.?!:;()\-]/ );

    # for chef
    next if($first_token =~ m/chef/i);
    next if($second_token =~ m/re/i);
    next if($second_token =~ m/released/i);

    # for puppet
    next if($first_token =~  m/rc/i);
    next if($first_token =~  m/release/i);
    next if($first_token =~ m/users/i);
    next if($first_token =~ m/available/i);
    next if($second_token =~ m/users/i);
    next if($second_token =~ m/announce/i);
    next if($second_token =~ m/available/i);
    next if($second_token =~ m/source/i);
    # increment
    $index++;
    last if ( $index > 20 );

    # output
    # print "$$tscore{ $bigram }\t"           .
    #        "$$bigram_count{ $bigram }\t"     .
       print "$bigram\t\n";

    }
print "\n";

# get tri-gram counts
my $trigram_count = $ngrams->trigram_count;

# process the first top 10 tri-grams
$index = 0;
print "Tri-grams (count, tri-gram)\n";
foreach my $trigram ( sort { $$trigram_count{ $b } <=> $$trigram_count{ $a } } keys %$

    # get the tokens of the bigram
    my ( $first_token, $second_token, $third_token ) = split / /, $trigram;

    # skip punctuation
    next if ( $first_token  =~ /[,.?!:;()\-]/ );
    next if ( $second_token =~ /[,.?!:;()\-]/ );
    next if ( $third_token  =~ /[,.?!:;()\-]/ );

    # for chef
     next if($first_token =~ m/chef/i);
     next if($second_token =~ m/re/i);
     next if($third_token =~ m/re/i);

    # for puppet
    next if($first_token =~ m/release/i);
    next if($first_token =~ m/rc/i);
    next if($second_token =~ m/users/i);
    next if($second_token =~ m/announce/i);
    next if($second_token =~ m/puppet/i);
    next if($third_token =~ m/team/i);


    # skip stopwords; results are often more interesting if these are commented out
    #next if ( $$stopwords{ $first_token } );
    #next if ( $$stopwords{ $second_token } );
    #next if ( $$stopwords{ $third_token } );

    # increment
    $index++;
     last if ( $index > 10 );

    # echo
    # print $$trigram_count{ $trigram }, "\t$trigram\n";
    print "$trigram\n";
    }
print "\n";

# get quad-gram counts
my $quadgram_count = $ngrams->quadgram_count;

# process the first top 10 tri-grams
$index = 0;
print "Quad-grams (count, quad-gram)\n";
foreach my $quadgram ( sort { $$quadgram_count{ $b } <=> $$quadgram_count{ $a } } keys

    # get the tokens of the bigram
    my ( $first_token, $second_token, $third_token, $fourth_token ) = split / /, $quad

    # skip punctuation
    next if ( $first_token  =~ /[,.?!:;()\-]/ );
    next if ( $second_token =~ /[,.?!:;()\-]/ );
    next if ( $third_token  =~ /[,.?!:;()\-]/ );
    next if ( $fourth_token =~ /[,.?!:;()\-]/ );

    # for puppet
    next if($first_token =~  m/rc/i);
    next if($first_token =~ m/users/i);
    next if($first_token =~ m/team/i);
    next if($second_token =~ m/users/i);
    next if($second_token =~ m/announce/i);
    next if($second_token =~ m/puppet/i);
    next if($second_token =~ m/announce/i);
    next if($third_token =~ m/team/i);
    next if($fourth_token =~ m/users/i);


    # skip stopwords; results are often more interesting if these are commented out
    #next if ( $$stopwords{ $first_token } );
    #next if ( $$stopwords{ $second_token } );
    #next if ( $$stopwords{ $third_token } );
    #next if ( $$stopwords{ $fourth_token } );


    # increment
    $index++;
    last if ( $index > 10 );

    # echo
    # print $$quadgram_count{ $quadgram }, "\t$quadgram\n";
    print "$quadgram\n";
    }
print "\n";

# done
exit;
```

Appendix C

Glossary

app: Application

SUS: System Usability Scale

SEQ: Single Ease Question

API: Application Program Interface

CSV: Comma separated value

SQL: Structured Query Language

mbox: Mail Box file

MySql: Database engine(service)

JSON: JavaScript Object Notation

PDF: Probability Density Function

CI: Confidence Interval

SSL: Secure Socket Layer

COPBL: CFEngine Community Open Promise-Body Library

OS: Operating System

OSAL: Operating System Abstraction Layer

DSL: Domain Specific Language

DHCP: Domain Host Control Protocol