International Journal of Computer Applications Technology and Research
Volume 11–Issue 07, 291-295, 2022, ISSN:-2319–8656
DOI:10.7753/IJCATR1107.1006

An Overview of Domain Details Tool

B Somu Sashank
Computer Science and Engineering
RV College of Engineering
Bangalore, India

Dr. Sandhya S
Computer Science and Engineering
RV College of Engineering
Bangalore, India

Abstract: The Get Domain Details uses the DIG command to provide hostname or Domain Name Server details for particular location or edge server IP. An information collection and preparing device is accounted for DIG. The tool permits the management of a product cycle to have the option to extract information from a repository to groom the information and to create reports. The reports are pointed toward helping the management in the control of the product advancement process. DIG allows the executives to construct a total portrayal of the efficiency information to more readily coax out the boundaries and figure out the reasons for variety in the information. Also, the information acquired and prepped by DIG might be utilized to align and apply process control models.

Keywords: software testing, report, GANTT plan, deployment, dnslookup, plugins

1. INTRODUCTION

This paper presents the utilization of the tool DIG. The tool was developed to satisfy the requirement for information collection in a software process environment; especially for the approval of a model of the interleaved occasions of coding and testing incremental programming improvement. DIG is a robust command-line tool developed by BIND for querying DNS nameservers. It can identify IP address records, record the query route as it obtains answers from an authoritative nameserver and diagnose other DNS problems. Dig is more advanced than dnslookup and host commands. It is noticed that such a tool could uphold numerous information-driven drives including management works, process improvement, and control, preparing prescient reproduction models or measurements development for different purposes. Generally speaking, the alignment of a product cycle simulation model demands tedious manual extraction, preparation, and translation of verifiable interaction execution information from different sources. In these assignments, the issues may not loan themselves to natural arrangements; for this situation, it could be useful for the model creators to give devices to plan specific process elements to demonstrate ideas, and for extricating model boundaries from defective arrangements of information. While DIG worked to align a particular model, its more extensive utility lies in the way that it gives a system for the development and translation of time-series information from process curios. This tool you can verify if the routing between a user and edge server is optimal and if the domain has any Canonical Name records, namely whether your domain has other domains acting as its aliases and if there are any issues with the resolution of domain names.

2. GOALS

When interpreting the data removed from the artifacts of a specific cycle, looking at the information inside the setting of the interaction that produced it is vital. For instance, think about Figure 1. In the figure, by overlooking the GANTT plan one could reason that the efficiency of code creation, drops off moderately immediately followed by a significant stretch of low efficiency. Given the appropriate setting, notwithstanding, one can without much of a stretch see that the store information for Release 0.2 addresses the execution of two undertakings: Coding and then the slow Rework task. This is the essential objective of the DIG instrument: to give the semi-computerized component for incorporating crude process information with the setting given in the task plan. Upon this objective we place three positive properties: i) The center application ought to uncover, in an object structure, the gathered information by means of a public connection point expected for those applications which would consume the information; ii) The revealing capability of the device ought to help a secluded 'estimation' which is free of a specific perspective on the information/computations, and iii) The tool ought to have the option to work with the momentum variants of the cycle relics and give changed information and computations consistently upon artifact update.

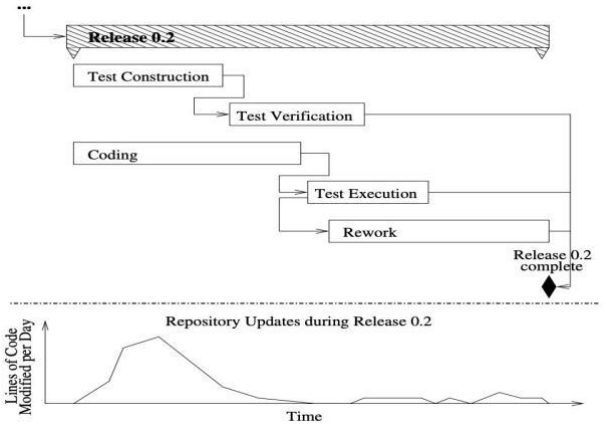

Figure. 1 Interpretive context provided by project schedule

3. RELATED WORK

The literature on automated data collection and analysis lays out the primary motivating factors driving the construction of automated tools. They may be briefly summarized as i) Data collection is expensive, and there is no instant gratification for doing it; ii) Data collection is unreliable, owing in no small part to the fact that developers find it irritating and secondary in priority; and iii) Errors in the recording of the data seem to occur more frequently in the critical parts of the development process;

precisely when the data are most needed. The literature espouses a particular modular architectural structure consisting of the following components : i) Data collection/Data grooming - those parts of the system which act as an interface between the data storage/processing central component and the raw data sources (e.g. SCM tools, Project Mgmt. tools, IDEs, etc.); ii) Data storage and representation - the central component which gathers the groomed data from the collection components, and makes it available in some object form to client applications/applets; and iii) Data clients - the client applications/applets which use the data provided by the data storage module to calculate metrics, charts, or model parameters. Here, the term 'component' is used to refer to the idea of 'add-on software components, both to a central application – as per our approach illustrated in Figure 2, as well as 'deployment modules' as per where the notion is that of a stand-alone client application which runs elsewhere on the network. The authors propose the idea that future tools should support "a more explicit view of the process," as the current tools tend to focus on the raw data from the sources in isolation from the perspective provided by the context of the process as a whole. Here we note the distinction between product and process metrics. The product variety may likely be calculated without regard to an explicit representation of the process; these are metrics such as complexity, size, etc. The process metrics, such as defect insertion rates, productivity, etc., require context for interpretation. For example, while one may simply use completion data to derive coarse productivity metrics, we note that extrapolation from such metrics must assume that future work is performed in a similar environment; it is only by considering the confounding environmental factors (e.g., vacation days, concurrency in the schedule, etc.) that one may derive an understanding of process metrics that is capable of being applied to widely varying future environments. We propose that there is value inherent in tying the raw data to the project schedule as the minimum satisfaction of the premise that the process should find representation in any such data collection tool. To support this conjecture, we draw an analogy with best practices using high-level programming languages: it is suggested that the 'goto' command be avoided because it destroys the logical Constructs which simplifies program

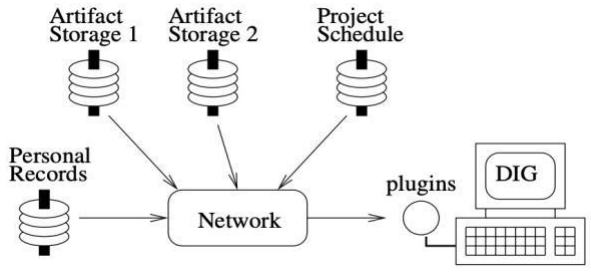

Figure. 2 Deployment of DIG in a networked environment

We make a similar case for why it's important to include raw time-series data in the project plan; this way, external effects and confounding factors can be easily identified and the link between the factors and the metrics under study may be better understood. To increase understanding, we (in DIG) compel the linkage of time-series data to schedule components.

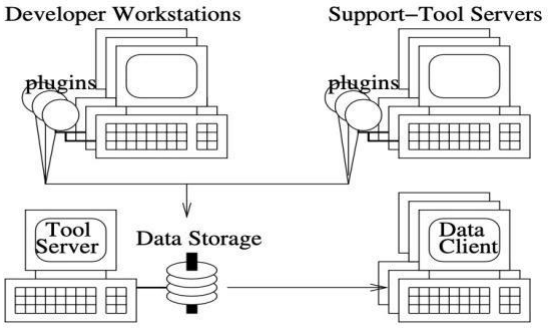

Figure. 3 Tool Deployment Scheme

4. ARCHITECTURE OF DIG

To promote wide applicability, DIG is constructed as a framework; a core application that manages the association of data and schedule elements, and a set of organization-specific plug-ins to parse out the data from the proprietary artifacts and provide. The ConcreteCompletionDataPlugin class in Figure 4 represents the custom-written parser; the classes represent the standardized interfaces to the raw data that the parsers must provide; the interface exposes a set of mappings (i.e. 'cross references'): i) from the Tasks to the Workforce members who were responsible for their completion; ii) from each Workforce member to those units of change of the Task which for which they were responsible, and iii) from each unit of change to a time-series of completion data. In Figure 5 we give an example of the mappings that were required to extract the data from the SCM system in our case study work. As can be seen, the SCM Completion Profile acts as a Facade for the calculations on the SCM Repository elements that are parsed out of the data within the SCM plug-in. For this example, The plugin's user interface requires the specification of the ChangeIDs associated with each Schedule Task of interest; where ChangeIDs are a part of our partner's change management system. This is an instance of semi-automated data collection, as the mappings must be supplied manually, but the subsequent analysis and computation are automated. In designing a general data representation for the schedule and raw completion data, we have chosen to provide an object structure that clients may traverse in order to gather the collected data. This structure is an object representation of a GANTT-like work-breakdown structure representing the schedule, with the plug-in-supplied data attached to the leaf tasks as seen in Figure 5, in the tree structure to the bottom-left of the diagram. The link to the plug-in-supplied data is represented by the association to the ConcreteCompletionProfile through the CompletionProfile interface. The calculation model of DIG is a point of novelty. In contrast to the common architecture seen in the literature, we provide an interface for the specification of a calculation independent of a view. Thus, one may implement a calculation once, and use it to drive several graphical views, textual report generators, more complex calculations, etc. In order to support the read-only, external usage of the current process artifacts, DIG and its plugins must identify data elements by immutable identifiers. Thus, upon the reloading of a project configuration after an artifact update, all of the artifacts are reparsed, and the mappings are reapplied – were still applicable – to the new set of artifact entities. DIG also supports a 'type' tag for the tasks in the schedule. Currently, DIG supports five task types: i) Feature Coding; ii) Test Case Authoring; iii) Test Case Verification; iv) Regression; and v) Defect Elimination (debugging); where 'Test Case Verification' refers to the preliminary execution of a test case against an internal product

release for the purpose of evaluating the correctness of the test case.

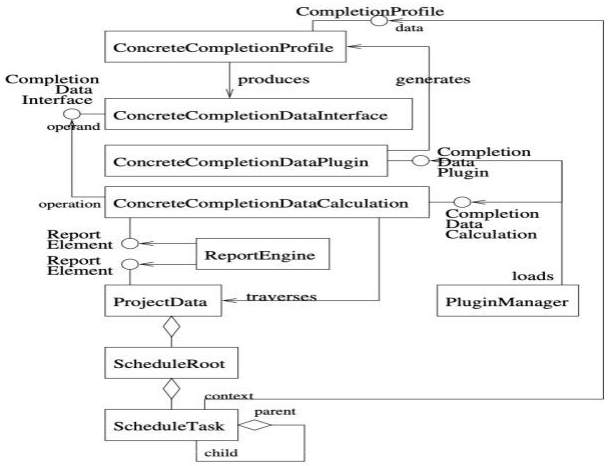

Figure. 4 Inter Object relations in DIG Framework

These categories are not intended to cover the entire spectrum of task types but are intended to demarcate certain tasks as being 'of interest' so that the majority of the schedule may be ignored were extraneous. Further, by having the task types, heuristic methods may be applied (e.g., for determining the flows of work between tasks)

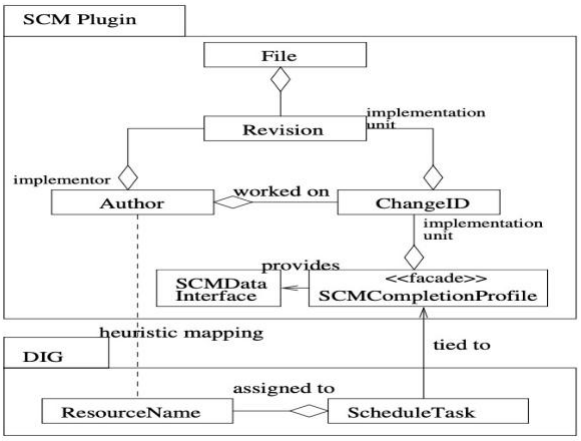

Figure. 5 Proprietary data manipulation via plugins

5. RESEARCH METHODOLOGY

The following discussion touches upon the interesting problems encountered during the construction of DIG, and their methods of resolution. In some cases, we note how automated methods might be facilitated given a small process augmentation.

Task completion. DIG was built originally to determine production rates and to characterize how the productivity rates change as a function of the proportion completed of tasks. Thus, one needs the notion of task completion, and more specifically for the characterization of productivity-rate change, one needs to be able to define the partial completion of a task. While being a seemingly benign requirement, the scheduled tasks Often represent the completion of activities (i.e., coding, testing, etc.) with respect to a particular feature (or set of features); which begs the question, "What does 'half of a feature' mean?" In our plugins, we assume that most of the partial completion is understood in terms of the proportion of the artifact that is complete (e.g., the proportion of the total lines of code being currently complete). The consequence is that we must wait for a feature to be completed before DIG can analyze its data. Also, consider the question of how defects and changes in requirements fit into the definition of task completion. We mark the completion of a task as the first point at which all of the known work for a task is complete. Future defects and changes to requirements are treated as reparative work.

Workforce allocation. Our conversations indicate a preference in the industry for allocating the workforce to tasks in a task-centric, rather than worker-centric, manner; ensuring that a task has the 'right' people outweighs consideration of the workload of the individual workers. This necessitates the notion of task concurrency for a worker. Our plugins treat the allocation of workers to tasks in terms of worker equivalents - which we define as the fractional portion of an average worker's effort which is applied to a task as the result of giving an equal portion of each work-day to each of the active concurrent tasks to which the worker is assigned. It should be noted that it is possible that all tasks to which a worker has been assigned are blocked due to unsatisfied dependencies in the schedule. In such a case, we assume that these workers are pulled into external projects and thus contribute nothing to the current project.

Process representation. The process representation used by DIG is the stripped-down entirely pragmatic version found in the inter-task dependencies in the schedule. Thus, DIG handles changes in procedure seamlessly: for those projects which must now conform to the new process, the schedule will be updated, and DIG will automatically use the new dependencies. For those projects which are allowed to use the older process (i.e., grandfather clause), there is no schedule change, and so the official procedure change is transparent to DIG.

Flow of work. The schedule alone may not be sufficient to understand the dynamics of the development and test process; as one team lead from our collaborative partner put it: "There're lots of little cycles that you just can't represent very well in a work-breakdown structure". This is per- haps the best illustration of what we term 'flow of work' – the quote above was made in reference to the cycles of rework and re-evaluation that are part of the debugging process. More technically, workflow refers to the producer/consumer relations between the tasks. One cannot interpret the productivity of a faster downstream task without considering that slower upstream tasks may be limiting its productivity by 'starving' the downstream process for work. Models which require this data may attempt to acquire it heuristically using the 'task type' tags and the inter-task dependencies in the schedule.

Definition of task types. The task type tags may not ap- ply directly to the tasks in the schedule. Consider a common process definition where the development team is responsible for both the writing of code and its informal unit testing where

the schedule includes separate tasks for coding and the informal unit test. Any completion data which is as- signed to the coding task will likely include the execution of the unit testing activity as well. We note that by requiring the scheduling granularity to match the type tag granularity, this becomes a non-issue.

Scheduled Vacations. DIG extracts scheduled vacation days for employees from MS Project schedules and provides APIs for date calculations. Ambitious plugins may adjust for vacations in their interpretation of historical raw data, or in future projections.

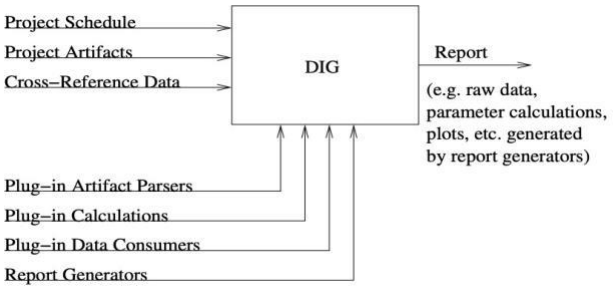

Figure. 6 Usage of the Dig tool

Relative task difficulty. DIG supports using the a priori initial work estimates from the schedule (in MS Project, these are the 'baselines') for calculating relative difficulty metrics. The motivation for this choice is a desire to include the expertise of the project planning team in the set of data that DIG can access.

Name mapping. The problem of disparity between the names of workers as they appear in one artifact vs. another. DIG provides a simple fuzzy name mapping API that compares the names by evaluating histograms of the letters A-Z present in the names in each artifact.

Ad-hoc artifact organization. For plug-ins that read collections of artifacts (e.g., test logs), obtaining the correct subset of the collection for a particular task may be nontrivial depending on the organization of the records. To handle ad hoc organization, the test log plug-in we implemented supports date filtering, file-name filtering, and compressed archive support so that one need not touch the archives to be able to read the appropriate subset of files.

Parameter correction. DIG provides the data and the context from which a simulator can be calibrated and executed. The modular calculation objects revises parameters estimates based on simulation accuracy, etc.

6. RESULTS AND DISCUSSION

The issues encountered in the construction of DIG lead to the following observations. The key to automating the entire system is traceability from the schedule to the elements in the artifacts. The majority of the functionality in DIG is to allow the specification of these cross-references. Implementation of a change management system, in contrast to separate change tracking and software configuration management, is a likely first step in building the process infrastructure to support automated model calibration. Integrating knowledge of the schedule into a change management system would be a final goal. Upon this information source, tools like DIG would become extremely simple to use. During initial testing of the DIG tool, we found multiple examples of improper semantics in the usage of the schedule dependencies – almost a colloquial dialect based on the originally intended semantics of the dependencies. Also, we found occurrences where dependencies were simply not used, and the desired temporal structure was imposed on the schedule by constraining the start dates of tasks to particular dates. While this still allows the normal mapping of data to tasks, it breaks any of the heuristic methods which rely on traversing the schedule dependencies. As a project lead in our collaborating company said, "No one wants to enter that data twice, you spend hours doing it just once". The colloquial usage of dependencies in the project schedule is a practice that should be avoided. While it may be an effective medium to convey the appropriate message to subordinate workforce members, it pre-includes the ability to use any standard tools to extract meaningful data from the schedule. Interest in the tool, the architecture, and potential applications may be directed to the authors via email.

7. FUTURE WORK

As the truism goes, "Programming is rarely finished"; while DIG is certainly not a business device, DIG is an exploration model valuable to handle control scientists who need to remove information from project storehouses with the end goal of boundary assessment and use in process reproduction models. Reports produced by DIG have likewise been thought of as valuable by test supervisors. A couple of inquiries stay as for the specific situation portrayal. It is widely known that an enormous number of gatherings during a day or week unfavorably influences productivity, so it appears to be that DIG ought to attempt to pull in a representation of "meeting thickness" to finish the image that we are attempting to draw around the relic information. As far as extension, DIG is right now custom-made to accumulate and work out time-series (for example efficiency) information. Future work ought to expect to extend the extent of DIG to envelop the full scope of information assortment tracked down in modern practice. The subject of "what is the fundamental arrangement of programmer information?" emerges in planning the information portrayal that DIG would have to help if seeking after this line; this might be a fascinating inquiry with regards to itself. Ultimately, for simplicity of organization, it appears to be that the construction of a "stock module library" of parsers for normal relics and computations for normal models/measurements/and so on would be of extraordinary guide.

8. ACKNOWLEDGEMENT

The Authors would like to extend their gratitude to Dr. Nagaraj G Cholli, Dr. Praveena T for their valuable comments and suggestions and providing the opportunity to study the development process, artifacts and records.

9. References

1. "Standard Glossary of Terms used in Software Testing", (PDF). Version 3.1. International Software Testing Qualifications Board. Retrieved, January 2018.
2. R. Abbas, Z. Sultan and S. N. Bhatti, "Comparative analysis of automated load testing tools: Apache JMeter Microsoft Visual Studio (TFS) LoadRunner Siege", 2017 International Conference on Communication Technologies (ComTech), pp. 39-44, 2017.
3. R. Khalid, "Towards an automated tool for software testing and analysis", 2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST), pp. 461-465, 2017.
4. Vasilyev, I. Paramonov and S. Averkiev, "Method and tools for automated end-to-end testing of applications for sailfish OS", 2017 20th Conference of Open Innovations Association (FRUCT), pp. 471-477, 2017.
5. Vasilyev, I. Paramonov and S. Averkiev, "Method and tools for automated end-to-end testing of applications for sailfish OS", 2017 20th Conference of Open Innovations Association (FRUCT), pp. 471-477, 2017.
6. Vishawjyoti and P. Gandhi, "A survey on prospects of automated software test case generation methods", 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 3867-3871, 2016.
7. P. Ammann and Offutt J, "Introduction to Software Testing" in Cambridge University Press., pp. 26, 2016, ISBN 9781316773123.
8. A Comparative Study of Automated Software Testing Tools Nazia Islam St.Cloud State University, 2016.
9. Cem Kaner, "(November 17 2006). Exploratory Testing (PDF)", Quality Assurance Institute Worldwide Annual Software Testing Conference. Orlando FL. Retrieved, November 22, 2014.
10. "Test Automation Tool comparison - HP", UFT/QTP vs. Selenium Aspire, Nov 2013.
11. W.L. Oberkampf and C.J. Roy, "Verification and Validation in Scientific Computing" in Cambridge University Press., pp. 154-5, 2010, ISBN 9781139491761.
12. M.G. Limaye, "Software Testing", Tata McGraw-Hill Education, pp. 108-11, 2009, ISBN 9780070139909.
13. K.A. Saleh, "Software Engineering", J. Ross Publishing, pp. 224-41, 2009, ISBN 9781932159943.
14. W. Wang, Y. Lei, S. Sampath, R. Kacker, R. Kuhn, and J. Lawrence, "A combinatorial approach to building navigation graphs for dynamic web applications," in Proc. of ICSM, 2009, pp. 211-220.
Mesbah, E. Bozdag, and A. v. Deursen, "Crawling AJAX by inferring user interface state changes," in Proc. of ICWE, 2008, pp. 122-134.
15. M.-C. D. Marneffe, B. Maccartney, and C. D. Manning, "Generating typed dependency parses from phrase structure parses," in Proc. of the LREC, 2006, pp. 449-454.
16. G. Little and R. C. Miller, "Translating keyword commands into executable code," in Proc. of UIST, 2006, pp. 135-144.
17. D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: helping to navigate the API jungle," in Proc. of PLDI, 2005, pp. 48-61.
18. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," ACM Comput. Surv., vol. 37, pp. 83-137, June 2005.

H. Liu and H. Lieberman, "Programmatic semantics for natural language interfaces," in Proc. of CHI Extended Abstracts on Human Factors in Computing Systems, 2005, pp. 1597-1600.