Regressions in the 21st Century – Tools for Global Surveillance

David Crutchfield
CAD Engr Sr Principal
Cypress Semiconductor
Lexington, KY 40507
daac@cypress.com

Tushar Grupta
CAD Engr
Cypress Semiconductor
Bangalore, India
tugu@cypress.com

Frank Roberts
CAD Engr Sr
Cypress Semiconductor
Lexington, KY 40507
jfro@cypress.com

Venkataramanan Srinivasan
CAD Engr Sr Staff
Cypress Semiconductor
Bangalore, India
vse@cypress.com

***Abstract*-Verification of highly configurable and reusable IP and systems often falls into the critical path of the development cycle. Higher scrutiny is placed on verification to provide information about status, coverage, trends, and path to closure. This leads to metrics gathering within and among projects throughout a given company to provide management pertinent data for making proper decisions with regards to resource allocation, hardware infrastructure, and license utilization. This paper describes a system built and utilized at Cypress Semiconductor to gather metrics across projects company-wide and make them available by anyone on demand. The paper will discuss how the metrics are gathered, where they are stored, and how they are accessed.**

I. Introduction

Shrinking technology nodes have caused complexity in semiconductor designs to increase substantially. At the same time, the time-to-market window is shrinking due to product delivery requirements. These trends create marketing and management challenges. Every activity in the design cycle must be as efficient as possible to meet the requirements of the business. Verification tasks have a pivotal role in design development cycles; ensuring quality silicon is delivered as quickly as possible. Verification often falls into the critical path and is scrutinized for status and progression throughout the process. An agile semiconductor company must track verification activities as closely as possible to ensure best in class verification process, resource management, compute infrastructure, and license management. These systems and processes buttress growing business needs and shrinking time to market.

In a multi–national company, verification teams may be scattered around the world. These verification teams are responsible for ensuring the quality of design projects for multiple business units, and running hundreds of regressions that contain thousands of test cases to cover a wide range of functionality. Without metrics gathering tools, verification engineers must manually process a huge amount of regression data, including logs, metrics, status, and coverage, in order to generate high-level reports, trends, and statistics for tracking verification efficiency. High-level reports, trends and data are necessary to provide proper feedback and drive continuous improvement. The amount of data becomes overwhelming and unmanageable as the number of products, regressions, and tests grow. What is needed is a centralized, web-based, verification monitoring and reporting system automated to provide

quick access to all process metrics to help answer any question that could arise while trying to improve the overall verification process.

II. Definition of Terms

JSON	JavaScript Object Notation – A text-based object description format, often used for passing data between programs.
LDAP	Lightweight Directory Access Protocol – Stores user information and handles authentication for most of Cypress' online systems.
LSF	Platform Load Sharing Facility – Workload management platform, job scheduler, for distributed HPC environments.
SQL	Structured Query Language – A language for specifying operations on a relational database.
UCDB	Unified Coverage DataBase – A single persistent form for various kinds of verification data, notably: coverage data of all kinds, and some other information useful for analyzing verification results. The UCDB is used natively within Questa SIM.
Questa VM	Questa Verification Management – Functionality within Questa SIM that offers a wide variety of features for managing the regression. These features are built upon the UCDB.
VMS	Verification Management System – Internally developed tool for gathering design and test bench file information, compilation arguments, simulation arguments and test information, launching each task, and collecting regression information and status.
Questa VRM	Questa Verification Run Management – Mentor's tool for launching verification tasks within a regression.

III. System

Fig. 1 highlights the entitled work environment for verification engineers within Cypress. Verification teams from around the globe are executing regressions on sub-projects that will eventually be used to build higher level systems including products for different customer applications. By storing information from those regressions in a centralized database, project progress can be followed from anywhere within Cypress. Ideally, any data can be accessed by authorized personnel through web queries. This data can be used to create charts and trends over time to ultimately make educated decisions about current project delivery and better position the company for future success through proper planning. This section discusses the make-up of the system.

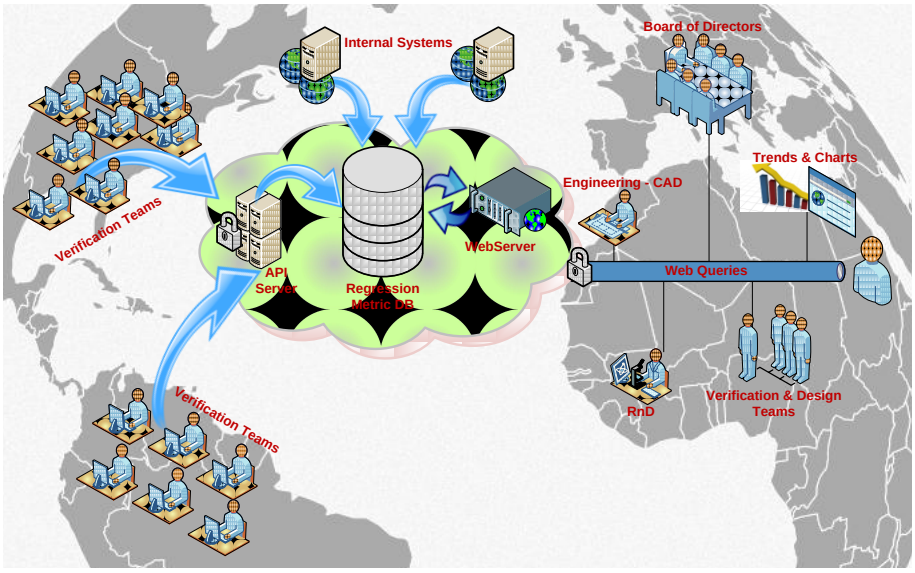

Figure 1: Verification Monitoring & Reporting Entitlement

A. *Existing Verification Framework*

Years ago Cypress developed VMS (Verification Management System) to manage logic verification regressions across the entire company. Along with the VMS tool, the project created a standard for design and test bench

organization, specification of tool arguments, test list creation, regression status reports, and coverage information. VMS leverages Mentor Graphics' Questa VRM (Verification Run Management) tool to launch compilation and simulation jobs through a load sharing facility (LSF), and to generate and merge functional coverage information.

Development of the existing system helped standardize the functional verification work flow, but it did not solve the problem of collecting and storing regression data. Tools to analyze and present regression data had been developed on a per-project basis, but regression data was not readily available to engineers or managers not already familiar with a particular project, and it was not easily accessible outside of the site where it was created. Presentation of the data was still a manual process, and required engineers to manually generate trend plots of coverage, pass/fail status, license usage, and hardware utilization for each project review. The process was very time consuming, and often fell in the critical path of development.

To address these problems VMS 2.0 was created to automatically gather and store important project metrics for each regression. The system gathers metrics to a central database, and provides a web interface that allows any employee in the company access to regression data and trend plots for every verification project.

B. Overview of VMS 2.0

VMS 2.0 consists of a web application backed by a database of regression metrics. This application is build on the Ruby on Rails web framework. Regressions executed through VMS have metrics automatically uploaded to the database after each regression. The regression database organizes regression data by project, and associates measurements of each metric with either a specific regression run or a specific simulation from that regression. In addition to runtime metrics, such as coverage, license usage, and CPU usage, VMS 2.0 integrates data from other Cypress systems to provide data related to project schedules and project defect tickets among other things.

The web interface allows users to browse projects and view a default set of trend plots, as well as raw data. Users may view data from the project level, the regression level, or at the level of individual tests. Users can generate custom trend plots of any regression metric, and may even plot multiple metrics on the same plot. Through the web interface users can set up a custom landing page, a "dashboard", that shows information only for projects they select.

Fig. 2 shows the architecture of VMS 2.0 and its various sources of information. Sources include the Verification Management System (VMS), the Project Management System, and the Defect Tracking System. Data gathered will be stored in the VMS 2.0 Database (VMS 2.0 DB), which is accessed through the VMS 2.0 Web Interface. VMS 2.0 utilizes Cypress LDAP servers to authenticate users, so that users may log into the VMS 2.0 system using the same credentials as they would for other Cypress systems.

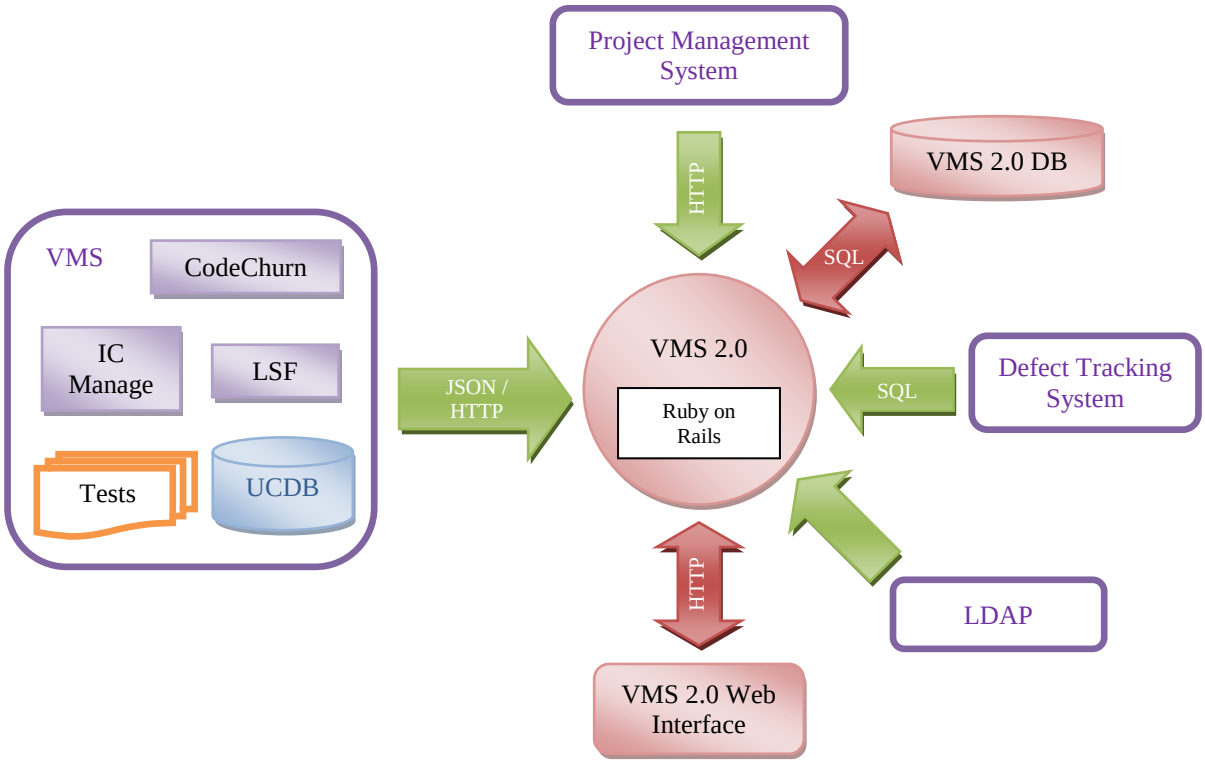

Figure 2: Application Architecture

C. Gathering Information

A major source of metric information is contained within a Unified Coverage Database (UCDB). This file is native to Mentor Graphics' Questa Simulator and is a single persistent form for various kinds of verification data, notably: coverage data of all kinds, and test specific metrics useful for analyzing and recreating verification results. A UCDB is generated for each individual test. In addition to standard metrics collected by Questa, VMS adds information important to track project progress. For instance, the amount of changed code of the IP each week is desirable. The overall picture of project closure is easier to see when coverage, number of passing and failing tests, and the amount of code changing can be seen together. If coverage is increasing, the number of failures is decreasing, and code is stabilizing, then project completion is believable. The opposite of any of these would push out project completion.

Other information is gathered from various sources, such as, LSF, IC Manage, project management system, and change and defect tracking system. Each of these is discussed below:

1. LSF – A parallel process within VMS actively monitors LSF for compute usage throughout the regression. Queries to LSF gather metrics in addition to job status, including max active jobs, max pending jobs, pending reason and wait time, and load statistics on the compute farm due to the regression. Data extracted or derived from LSF is added to the regression level UCDB.
2. IC Manage– Cypress uses IC Manage for its IP repository. VMS obtains variant and configuration information for the project being verified from IC Manage and stores it in the regression level UCDB. VMS 2.0 uses this to tie data to a project revision.
3. Project Management System – An in-house web system that facilitates in managing design projects, tracking planned and forecasted milestones, provides up-to-date project status like active, hold, deleted, etc. It also provides information about key people involved in a design project, such as, chip lead and verification lead.
4. Defect Tracking – An in-house developed web system that helps engineers track each defect identified by cross-functional teams on a design throughout the company. This system provides information about how many bugs are identified by a verification team on a particular design project and how many have been closed by design teams on a weekly basis. Management teams from both design and verification use this information to trend defect closure.
5. LDAP – Cypress uses LDAP to authenticate user access for all systems. By using LDAP VMS 2.0 allows users to use the same login credentials as other Cypress applications. Additionally, user customized dashboards are provided based on user login. Modifications to certain project information are limited based on differing user privileges.

Once a full regression has completed a collection script within VMS is launched to gather desired metrics from all sources that it has access to. Additionally, VMS 2.0 queries the Project Management and Defect Tracking Systems for Cypress specific project information. This is depicted in Fig 3. The collection script creates a regression object, extracts data from each source through different methods specific to each source, and updates the regression object with found data. Finally it stores the regression object to a file. An interface script loads the regression object, converts it to a JavaScript Object Notation (JSON) object, and sends the new object to the VMS 2.0 system using a JSON-based API through the HTTP protocol.

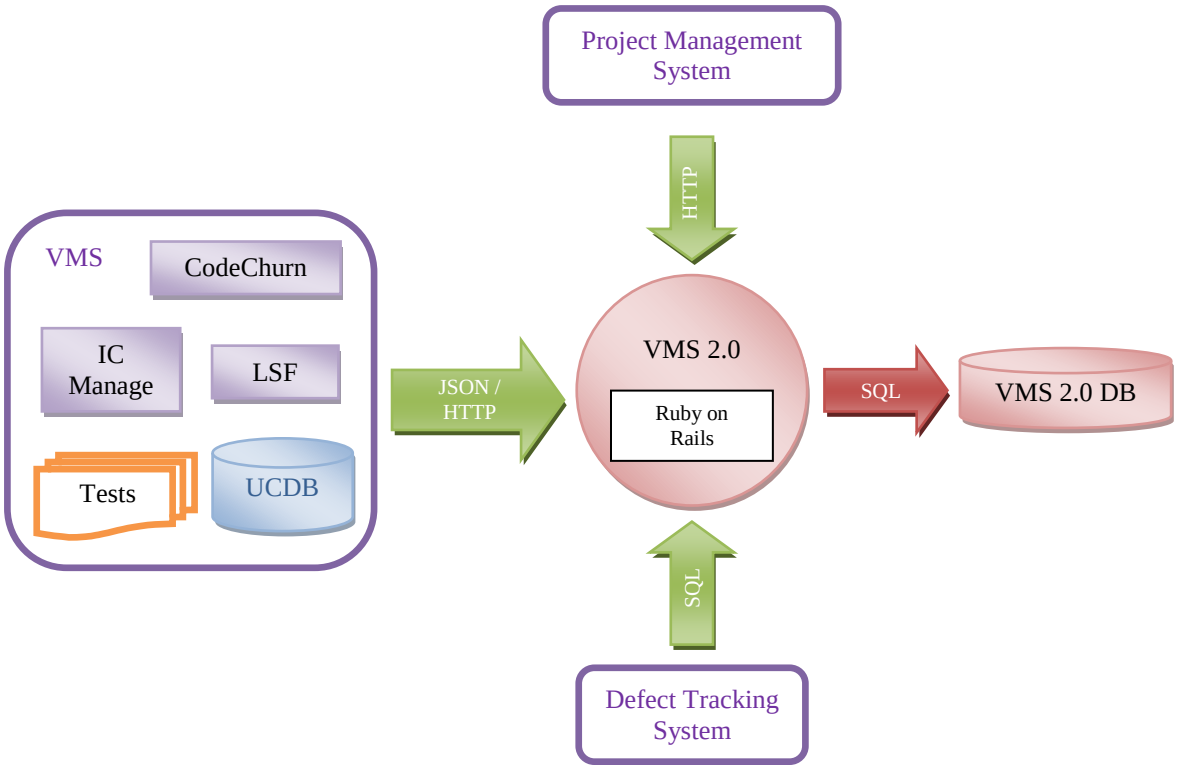

Figure 3: Data Sources for VMS 2.0

Below are two tables listing metrics gathered by VMS and uploaded to the regression database for extraction when needed. Table 1 includes regression level metrics. Table 2 provides metrics gathered for individual tests.

Table 1: Regression Level Metrics

Metric	Description
# passing	Number of passing tests per regression
# failing	Number of failing tests per regression
# new passing	Number of newly passing tests vs. the last run
# new failing	Number of newly failing tests vs. the last run
# tests warning	Number of tests with warning status
# tests unknown	Number of tests with an unknown status
# tests new warning	Number of new tests with warning status vs. the last run
# tests new unknown	Number of new tests with unknown status vs. the last run
# of tests	Total number of tests contained in this regression
Functional coverage	Total functional coverage achieved in this regression
Assertion coverage	Total assertion coverage achieved in this regression
Code coverage	Total code coverage achieved in this regression
Total coverage	Total combined coverage of all coverage types achieved in this regression
# unique tests	Total number of unique tests (no seed replication)
Number of tests	Total number of tests including all seeds
RTL codechurn	Amount of changed code of the RTL within the last week
TB codechurn	Amount of changed code of the test bench within the last week
Regression comments	Comments stored for this regression
Run time	Total run time of the regression

ICM workspace configuration	IC Manage configuration for the workspace
CPU hours per regression	Total number of CPU hours consumed by the regression
License-seconds per regression	Total number of seconds licenses were consumed for the regression
Average concurrent licenses per regression	Average number of concurrent licenses for the regression
Max number concurrent licenses per regression	Maximum number of concurrent licenses for the regression
Average CPU hours per project	Average regression time in CPU hours for the regression
Wall clock time per-regression	Total wall clock time for the regression
Set of hosts used per-regression	List of hosts used from LSF for the regression

Table 2: Test Level Metrics

Metric	Description
Functional coverage	Total functional coverage achieved for a test
Assertion coverage	Total assertion coverage achieved for a test
Code coverage	Total code coverage achieved for a test
Total coverage	Total combined coverage of all coverage types achieved for a test
Simulation mode	Simulation mode for a test
Test completion status	Status of test at completion time
Run time	Total run time for a test
Launch time	Launch time on LSF for a test
Completion time	Completion time of a test
CPU hours per test	Number of CPU hours consumed by a test
Average per-test time waiting on resources	Average amount of time a test waited on resource availability
Virtual memory per test	Amount of virtual memory allocated for a test
Memory consumed per test	Actual memory consumed by a test
Host used by each test	LSF host used by a test
Elaboration time per-test	Total simulation elaboration time for a test
Pending time per-test	Total pending time for a test

D. Storing Information

The JSON data sent from the collection script is received in the server and stored as a hash of information. In this implementation the hash has data stored at both the regression level as well as test level. Within the regression hash, individual test hashes are stored containing each test level item. Once the server receives this hash, an API is called to create the regression and test level objects in the database and store the corresponding data at the appropriate level. Additionally, links are created from each test to the regression database. Similarly, it links the regression to the project under which it has been uploaded. This database is constantly updated with project information throughout the company as regressions are completed. Once a new project is created, an automated email is sent to the project lead requesting additional input with regards to roles associated with the project. Fig 4. provides a simple overview of a project and its regression hashes. In this case Project1 has 1 through N regression hashes. Each regression hash has 1 through N tests. For each test hash, test code coverage, simulation time, and CPU time are shown as elements of the hash. In a real regression all elements in Tables 1 and 2 would be included for each test. These metrics may be accessed and plotted as needed.

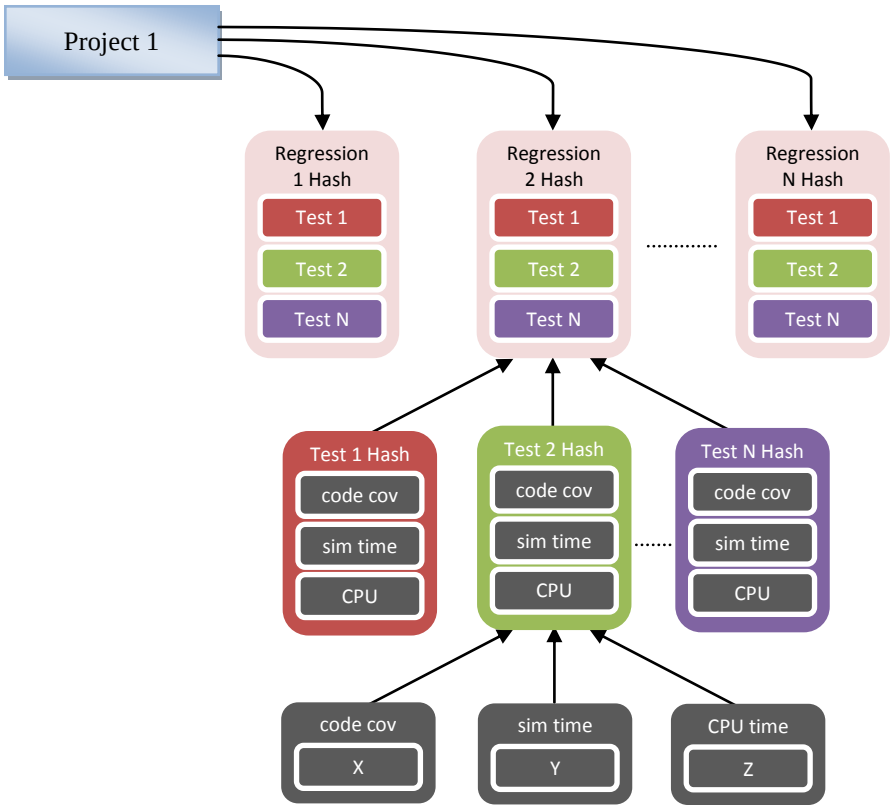

Figure 4: Simple Hash Representation

E. Extracting Information

Users access VMS 2.0 through the web interface. The VMS 2.0 web interface is built using Ruby on Rails. Ruby on Rails is a web development framework that implements a Model-View-Controller (MVC) paradigm. Fig. 5 provides a high level representation of the relationship. A model is a software representation of a database table. In VMS 2.0, Projects, Regressions, Tests, and Users all have models. The model represents the structure of the data associated with each of these objects. A view is a web page template that Ruby on Rails fills in with information from the models and then sends to a client web browser. A controller is the code responsible for processing model information so that it is ready to insert into a view.

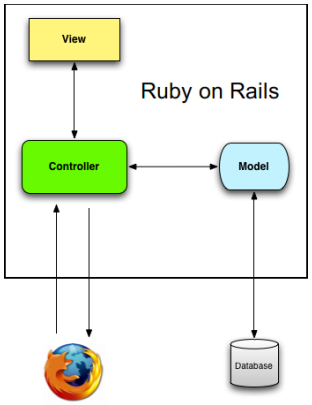

Figure 5: MVC Architecture of Ruby on Rails

When a user requests a certain page from VMS 2.0 (via a web browser), say "project_1/regressions/RTL_reg_50", Ruby on Rails routes the request to the Regressions controller, and passes along the project name "project_1" and the regression name, "RTL_reg_50". The Regressions controller instantiates a project model (for project_1) and a regression model (for RTL_reg_50). Ruby on Rails automatically retrieves data for these two models from the database. The Regressions controller then loads the single regression view (a template page for showing information on a single regression), and renders the view based on information from the project_1 and RTL_reg_50 models. The final rendered version of the view is in HTML and Javascript, which can both be parsed and displayed by the web browser. Ruby on Rails sends the rendered view back to the web browser to complete the request.

IV. Presentation

Once data has been gathered and stored in the database it is accessible throughout the company using the VMS 2.0 Web Interface as shown in Fig. 6. Through HTTP queries VMS 2.0 is directed to extract certain information from the database.

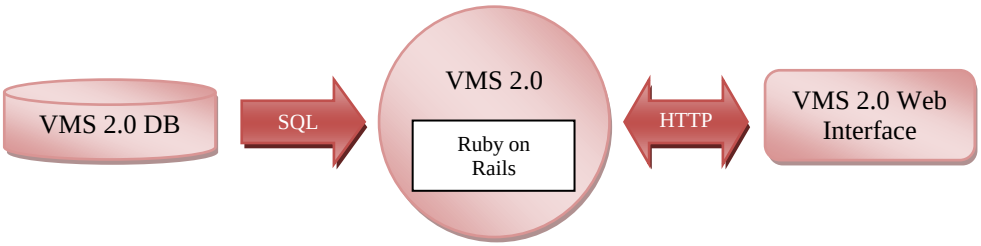

Figure 6: VMS 2.0 Database Access

Once a user has logged in, their User Dashboard will be displayed and all projects they have subscribed to are visible. Users can subscribe or unsubscribe to projects as desired. Through LDAP authentication users may be limited to exactly what they have permission to view. Fig. 7 shows an example User Dashboard. Here Code Monkey has three projects that he has subscribed to and they are listed as project_1, project_2 and project_3. Milestone dates and progress are indicated as IPSX columns.

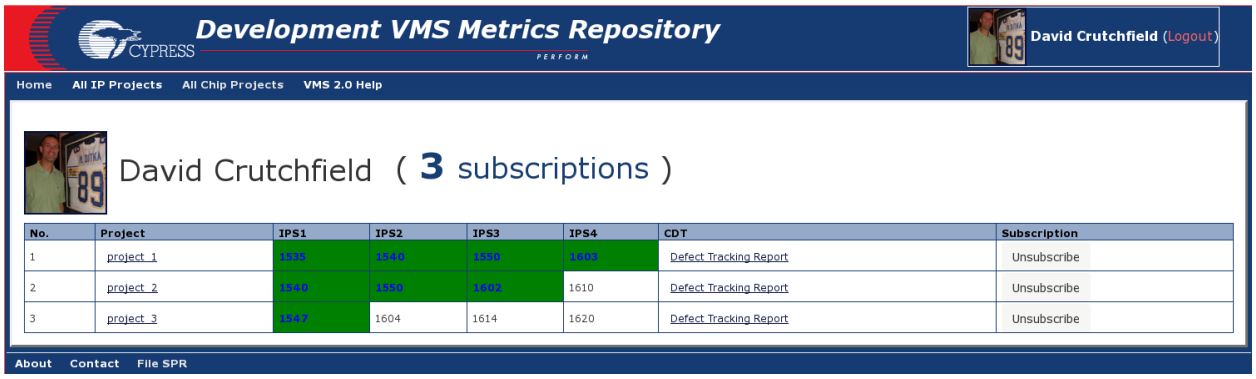

Figure 7: Initial User Dashboard

The tab "All IP Projects" is a link that the user can select to see all currently tracked projects in VMS 2.0. This is shown in Fig. 8. For each of these projects, if desired, the user can enable subscription by selecting "Subscribe" for a given project. By doing so, enables the selected project to be listed by default on the User Dashboard. The columns for each project listed are the same as that of the User Dashboard.

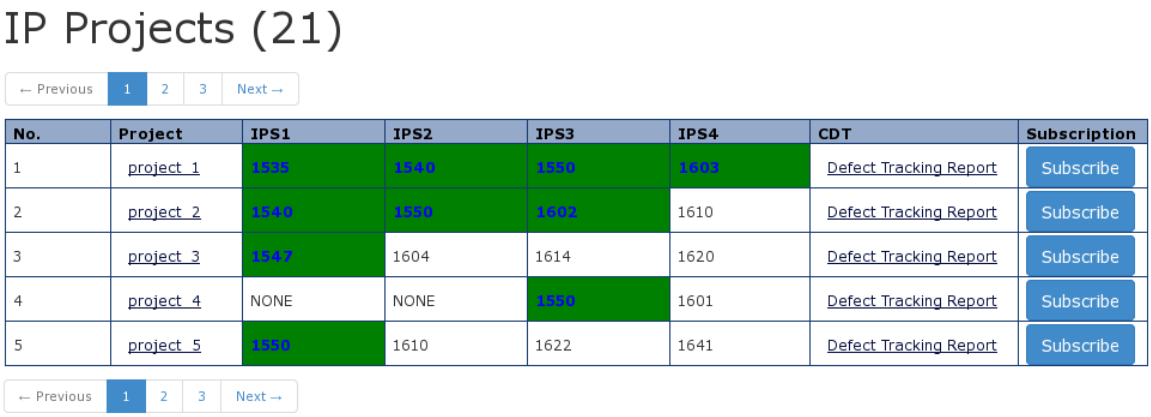

Figure 8: IP Project Dashboard

From the User Dashboard in Fig. 7 there is another tab, "All Chip Projects", that allows the user to see products that are being tracked through VMS 2.0. This is shown in Fig. 9. As with the IP Project Dashboard a user can select which products they would like to subscribe to. Once selected, each subscribed product will automatically be provided in the User Dashboard when they return to that page.

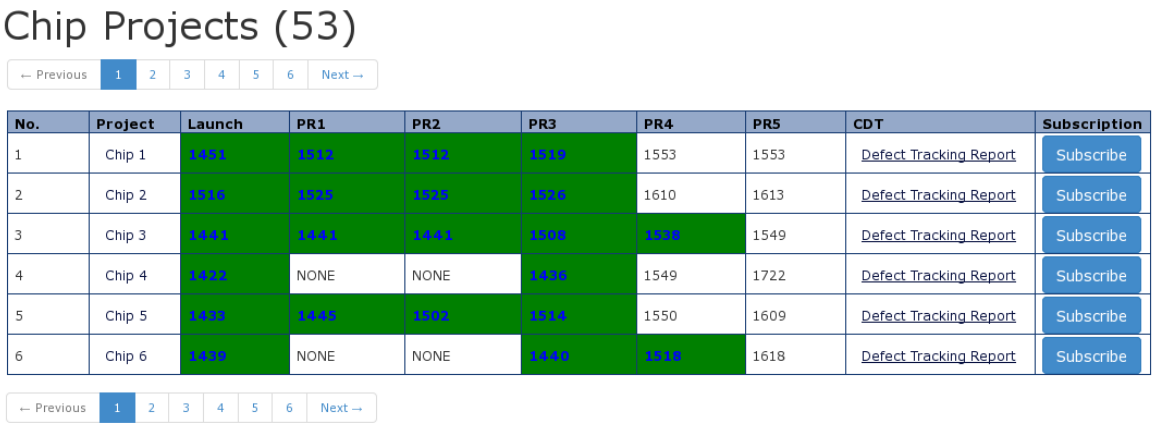

Figure 9: Chip Project Dashboard

In any of the project dashboards a user can select links provided for each project or product. Selecting the link routes the user to regression information for the selected project. Fig. 10 shows results for Project 1 listed in the IP Project Dashboard. At this point VMS 2.0 has stored fifty regressions for this project. Default columns of this dashboard are Total Coverage, Total Tests, Max Concurrent Jobs, Max Concurrent Licenses, Design Churn, TB Churn, Config Used, Total Regression Time, Work Week of Upload, and Comments. As can be seen, RTL_reg_46 has 131 tests and generated 98.65% Total Coverage. RTL_reg_47 added two more tests and generated 99.36% Total Coverage.

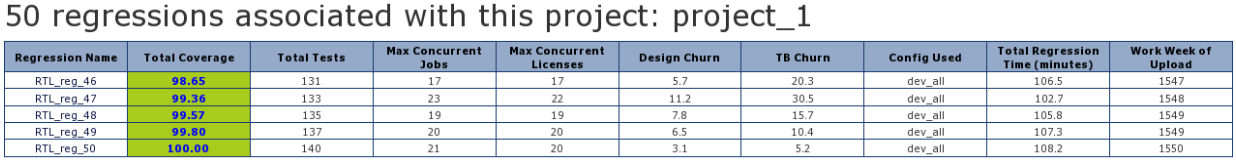

Figure 10: Project 1 Regression Dashboard

On the same web page as the Regression Dashboard a default plot of passing tests, coverage and regression run times over the life of the project is given. Fig. 11 presents the plot. In this example 50 regressions were executed to close coverage of 100%. Regression run times were over 100 min at the last data point.

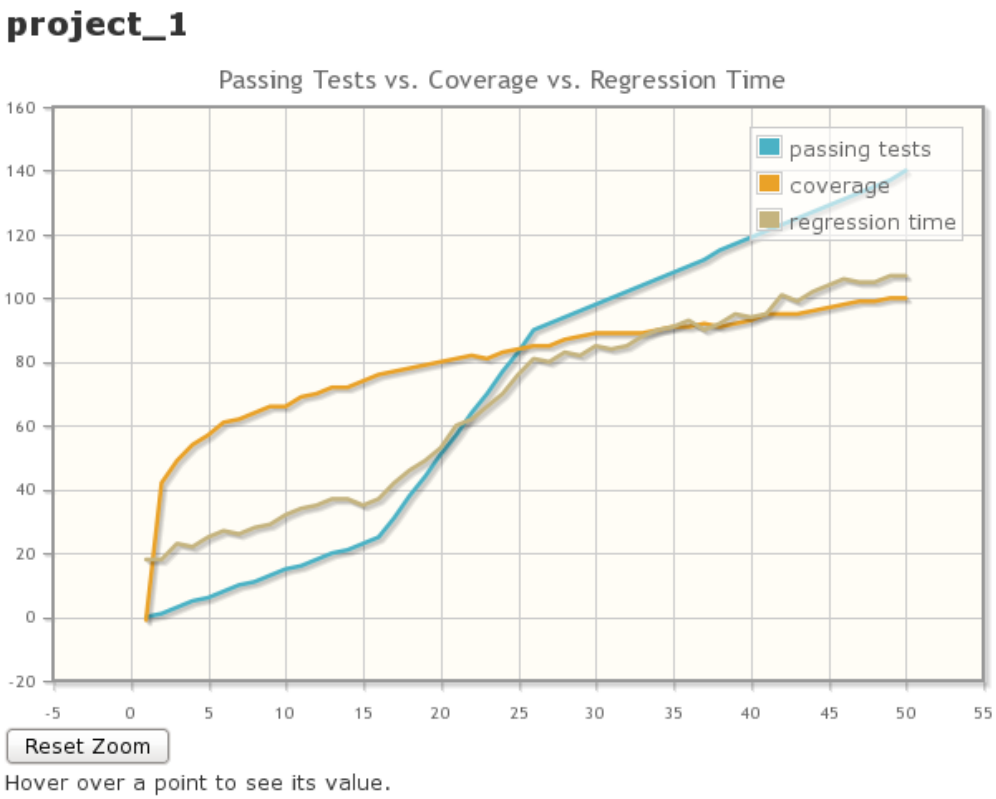

Figure 11: Project 1 Regression History Plot

Each regression is listed as a link and can be followed to get test information. Selecting the link RTL_reg_1 opens the dashboard shown in Fig. 12. In this dashboard each test for the regression is provided along with test specific metrics. Currently listed metrics are Test Mode, Status, Seed, Elab Time, Sim Time, Sim Memory, Sim V.Memory, LSF ID, Host, CPU Time, Real Time, and Wait Time.

5 tests associated with this regression: RTL_reg_1

Test Name	Test Mode	Status	Seed	Elab Time	Sim Time	Sim Memory	Sim V.Memory	LSF ID	Host	CPU Time	Real Time	Wait Time
basic_io_test	RTL	Pass	52577	35.2	127.5	235M	980M	43824	compute1	135	160.2	0.7
smoke_test_1	RTL	Pass	47322	47.5	98.4	130M	514M	43825	compute2	100	147	0.5
smoke_test_2	RTL	Pass	121815	42.5	90.6	154M	622M	43826	compute1	97.8	139.1	165.1
branch_rw_1	RTL	Pass	921959	87.4	195.4	347M	1138M	43827	compute2	257.6	288.4	150.2
rand_mem_test	RTL	Pass	20732	62.1	135.7	154M	622M	43828	compute5	178.5	201.3	307.2

Figure 12: RTL_reg_1 Test Dashboard

V. USAGE

Regression metrics and historical data are used in several ways at Cypress. Some of the use cases are listed in Table 3. The main goals for any historical data collection are to deliver projects predicatively on time and properly set customer expectations. Here usage is divided into three main categories. First, during project planning, historical data is important in accurately predicting what the development cycle will be for comparable projects. By looking at calendar time, the number of man weeks of effort (how many engineers for how long), license consumption over time, and hardware consumption, project managers and leads make informed decisions when allocating valuable resources. They can take this data and determine if the number of licenses will suffice, if better license allocation among projects would help, if more hardware is needed, or if more engineers are needed to meet certain targets. Knowing which knobs to turn based on historical data can be invaluable to setting and meeting customer expectations.

Second, during a project it is important to measure resource usage. Data being tracked includes the number of engineers contributing, licenses consumption, hardware consumption, and hardware efficiency. As stated with project planning, it is important to know which knobs to turn when trying to control schedule. If a project schedule is slipping due to license shortages, hardware issues, or other resource constraints then action can be taken early if regression metrics are readily available.

Finally, throughout project development, multiple status meetings with verification and design teams take place within Cypress. In the past verification engineers have been required to provide weekly status memos identifying the number of new tests created, how many tests are passing and failing, how many planned tests have not been completed, and what the current attained coverage is. In addition to weekly status meetings data was required for major milestones. For each of these data was manually collected by project leads and used to create trends. This step is all but eliminated with VMS 2.0. Data is always available for project leads or management to view and extract as needed. Furthermore, meetings that are required are shortened as ambiguity can no longer be argued when raw data is readily available.

Table 3: Metrics/Regression Data Usage

Usage Category	Metrics	Entitlement
Project Planning	Past project performance: calendar time	Accurately predict the development cycle based on past performance
		Set and meet customer expectations
	Resource usage: man weeks of effort license consumption hardware consumption	Understand what knobs to turn to improve future performance: number of licenses license allocation more hardware hardware distribution across sites more engineers
Current Projects in Progress	Resource usage: Number of engineers License consumption Hardware consumption Hardware efficiency	Understand what knobs to turn to improve current performance: get more licenses better license allocation more hardware is hardware working properly is hardware fully utilized more people needed
Status Meetings	Rate of progress: New test development Tests passing Tests failing Tests yet to be completed Coverage closure trend	Accurately track when a project will close
		Validate predicted milestones
		Eliminate weekly status documentation (memos)
		Reduce data collection at major milestones (dashboards are automated)
		Minimize required status meetings with entire team (data is always available)
		Shorter meetings (data is king, no ambiguity)

VI. Conclusion

Cypress historically has struggled to take a data-based approach to verification project planning. Data from previous verification projects was spread out across multiple projects on multiple continents, and no means for making the data readily available to the organization as a whole. Even when access to regression metrics was available, no standard set of analysis methods or presentation formats existed, making the data difficult to interpret consistently.

VMS 2.0 solves these problems. By integrating with the verification run management tool (VMS), regression metrics can be collected to a single, central database. The VMS 2.0 web interface provides a consistent presentation format for the data, and the plotting functionality provides the same set of analysis tools to everyone that uses the application. VMS 2.0 is accessible by anyone with a Cypress LDAP login and access to a computer on the Cypress network, which includes nearly every single employee.

VMS 2.0 is a critical tool in the verification planning process. VMS 2.0 allows project planners to be better informed when making decisions about the time and resources required to complete a verification project. The ability to track progress on verification projects is much improved, as it is no longer a manual process. Management can access up to date information about project progress at any time, for any project, without requiring interaction from the project owner. Tracking and analyzing resource usage for each project, enables better decisions when adjusting resource allocations, and provides a better understanding of how the adjustment may impact project schedules.

The intent in this paper is not only to show how a particular problem was solved within the Cypress verification organization, but also to provide a model that other organizations can use to solve similar problems. This paper provides a high-level description of the problem and a specific solution. In the interest of assisting others to implement similar systems, we have included a more detailed description of the Ruby on Rails application design in the appendices to this paper. These descriptions, in combination with a good Ruby on Rails tutorial (Michael Hartl's at www.railstutorial.org recommended), should provide one with a good starting point for developing a similar application.

Appendix A – Ruby On Rails Models

Although this appendix includes a basic description of how to interpret the data it contains, it will be much easier to understand with the context of a Ruby on Rails tutorial (try the one at www.railstutorial.org). We recommend that you at least read section 1.3.3, which explains the Model-View-Controller (MVC) paradigm, before reading this appendix.

A model is the Ruby (software) representation of a table in a database. The model includes all of the columns in the associated table (fields), as well as information on relationships that an entry in the table has with entries in other tables.

1) Project
   Fields: name, ips1, ips2, ips3, ips4, latest_cdt_report
   Relationships: Has many regressions, has many CDT reports, may belong to a parent IP, has 0 or more subordinate IP.

2) Regression
   Fields: name/tag, func_cov, assert_cov, code_cov, total_cov, # pass, # fail, # warning, # unique tests, # new pass, # new fail, # new warning, # contributing, # non-contributing, # with random seed, code_churn, ws_config, comments, CPU hours, licenses used, hosts used, license seconds
   Relationships: belongs to a project, has many tests

3) Test
   Fields: name, mode, seed, status, LSF id, LSF host, cpu time, real time, sim time, sim vmem, sim mem, completion time, time spent waiting on resources, elaboration time
   Relationships: belongs to a regression

4) User
   Fields: name, role, initials
   Relationships: has many projects, has many regressions, has many custom views, has one preferred view, has many custom plots

Appendix B – Ruby on Rails Controllers

Although this appendix includes a basic description of how to interpret the data it contains, it will be much easier to understand with the context of a Ruby on Rails tutorial (try the one at www.railstutorial.org). We recommend that you at least read section 1.3.3, which explains the Model-View-Controller (MVC) paradigm, before reading this appendix.

Controllers are the "brains" of a Ruby on Rails application. A controller consists of a set of methods, each of which knows how to assemble the information necessary to render a certain page view. This appendix provides high level descriptions of the methods implemented for each controller in VMS 2.0. In this context, a word prefixed with the "@" symbol indicates the name of a Ruby instance variable. For example, @projects may refer to the list of projects that a controller pulls from the database to satisfy a particular web request.

1) UsersController Methods:
   - index: Redirect to the user's preferred dashboard page.
   - show: Retrieve specified user data and set @user for the view.
   - new: Set @user to a new instance of the User model.
   - create:
     - Attempt LDAP lookup for user
     - If lookup is successful, create new user instance and save it to the DB.
     - If the save is successful, sign in and redirect the user to the default user dashboard.
     - If either the lookup or the save is unsuccessful, redirect the user to the default index page.

2) ProjectsController Methods:
   - index: Retrieve all projects. Set @projects for projects index view.
   - show: Retrieve the specified project. Set @project for the Project summary page.
   - new: Set @project to a new instance of the Project model.
   - edit: Retrieve the specified project. Set @project for the project edit view.
   - create: Create a new project instance, fill in info, and save the project to the DB.
   - update: Retrieve the project, fill in changed info, and save the project o the DB.
   - destroy: Remove the specified project from the DB (requires authentication).

3) RegressionsController Methods:
   - index: Retrieve recent regressions. Set @regressions for the regressions index page.
   - show: Retrieve the specified regression. Set @regression for the regression summary view.
   - destroy: Remove the specified regression from the database. (requires authentication)

4) TestsController Methods:
   - index: Retrieve recent tests. Set @tests for the tests index page.
   - show: Retrieve the specified test. Set @test for the test summary page.
   - destroy: Remove the specified test from the database. (requires authentication)

5) PlotsController
   - index: redirect to create
   - new: Set @custom_plot to a new plot object for the custom plot creation page.
   - create: Save the new custom plot to the database.
   - show: Set @custom_plot for use by the custom plot rendering page.
   - edit: Set @custom_plot to a new plot object for the custom plot editing view.
   - update: Update the selected custom plot with new information.
   - destroy: Delete the selected custom plot object.

Appendix C – JSON API Controllers

Although this appendix includes a basic description of how to interpret the data it contains, it will be much easier to understand with the context of a Ruby on Rails tutorial (try the one at www.railstutorial.org). We recommend that you at least read section 1.3.3, which explains the Model-View-Controller (MVC) paradigm, before reading this appendix.

In addition to the standard web-browser interface, VMS 2.0 provides a JSON/HTTP API to allow other scripts and applications to access information from the VMS 2.0 database. The API is implemented as part of the Ruby on Rails application, but has a separate set of controllers from the web interface. The most important difference between these controllers and the web interface controllers is that these controllers convert the data into JSON to send to the client, instead of using a Ruby on Rails view to render HTML.

1) ProjectsController
   create: Instantiate a new project object with supplied information. Save to the DB.
   show: Retrieve the specified project and render it to JSON.
   update: Retrieve the specified project object, add/replace data, and save it to the database.
   lock: Disallow updates by unauthenticated API sessions.
   unlock: Allow updates by unauthenticated API sessions (requires authentication).
   destroy: Delete the specified project object (requires authentication).

2) RegressionsController
   create: Instantiate a new regression object with supplied information. Save to the database.
   show: Retrieve the specified regression object and render it to JSON.
   update: Retrieve the specified regression object, add/replace data, and save it to the database.
   lock: Disallow updates by unauthenticated API sessions.
   unlock: Allow updates by unauthenticated API sessions. (requires authentication)
   destroy: Delete the specified regression object (requires authentication, must be admin to remove regression with tests).

3) TestsController
   create: Instantiate a new test object with supplied information. Save to the database.
   show: Retrieve the specified test object and render it to JSON.
   update: Retrieve the specified test object, add/replace data, and save it to the database.
   lock: Disallow updates by unauthenticated API sessions.
   unlock: Allow updates by unauthenticated API sessions (requires authentication).
   destroy: Delete the specified test object (requires authentication).

4) UsersController
   create: Instantiate a new user object with the supplied information. Fails if LDAP lookup is not successful (requires authentication and admin).
   show: Retrieve the specified user object and render it to JSON.
   update: Retrieve the specified user object, add/replace data, and save it to the database (requires authentication).
   destroy: Delete the specified user object (requires authentication and ownership or admin).

Appendix D – Ruby on Rails Views

Although this appendix includes a basic description of how to interpret the data it contains, it will be much easier to understand with the context of a Ruby on Rails tutorial (try the one at www.railstutorial.org). We recommend that you at least read section 1.3.3, which explains the Model-View-Controller (MVC) paradigm, before reading this appendix.

Each Ruby on Rails model has a set of associated views, that is, Ruby template files that generate HTML for a web page.

1) Project
   index: Lists all available projects.
   summary/show: Displays the project summary page for this project.
   edit: View/edit project information.

2) Regression
   index: Shows a list of recent regressions on all projects.
   summary/show: Displays the regression summary page for this regression.

3) Test
   index: Shows a list of recently completed tests.
   summary/show: Displays details for this test.