

Automation of SoC Verification Environment

Case study

Jukka Heikkilä

Applications Engineer, Verification Group, Synopsys

May 2019



CONFIDENTIAL INFORMATION

The information contained in this presentation is the confidential and proprietary information of Synopsys. You are not permitted to disseminate or use any of the information provided to you in this presentation outside of Synopsys without prior written authorization.

IMPORTANT NOTICE

In the event information in this presentation reflects Synopsys' future plans, such plans are as of the date of this presentation and are subject to change. Synopsys is not obligated to update this presentation or develop the products with the features and functionality discussed in this presentation. Additionally, Synopsys' services and products may only be offered and purchased pursuant to an authorized quote and purchase order or a mutually agreed upon written contract with Synopsys.

Agenda

- **Introduction**
- SW development practices
- Building continuous regression flow
- Example setups
- Summary

Introduction

Verification challenges for today's large IP/SoC teams:

- Planning the verification efficiently
 - Will all the product requirements get verified?
 - How to get the most important features verified first?
- Long verification times
- Complex verification environments prone to human errors
- Utilizing R&D infra efficiently (grid, multicore support)
- Monitoring progress in multi-site teams
- Keeping SoC design and related SW in sync

Well planned, scalable and continuous regression flow
with powerfull engines and automation is needed

Agenda

- Introduction
- **SW development practices**
- Building continuous regression flow
- Example setups
- Summary

Differences between HW and SW development

- SW verification is faster and more straightforward compared to HW verification
 - No need to worry about individual clock cycles
 - No infinite parallelism
 - No synthesis issues, netlist simulations, power simulations (UPF)
 - Test can take only few seconds
- HW processes follows typically tightly specified processes (to avoid expensive re-spins)
 - IP/SoC level functional verification can already take from days to weeks
- SW development processes are advanced and keep evolving all the time
 - Linear waterfall model → Flexible Agile model
 - DevOps (Continuous Integration, Continuous delivery, containers)

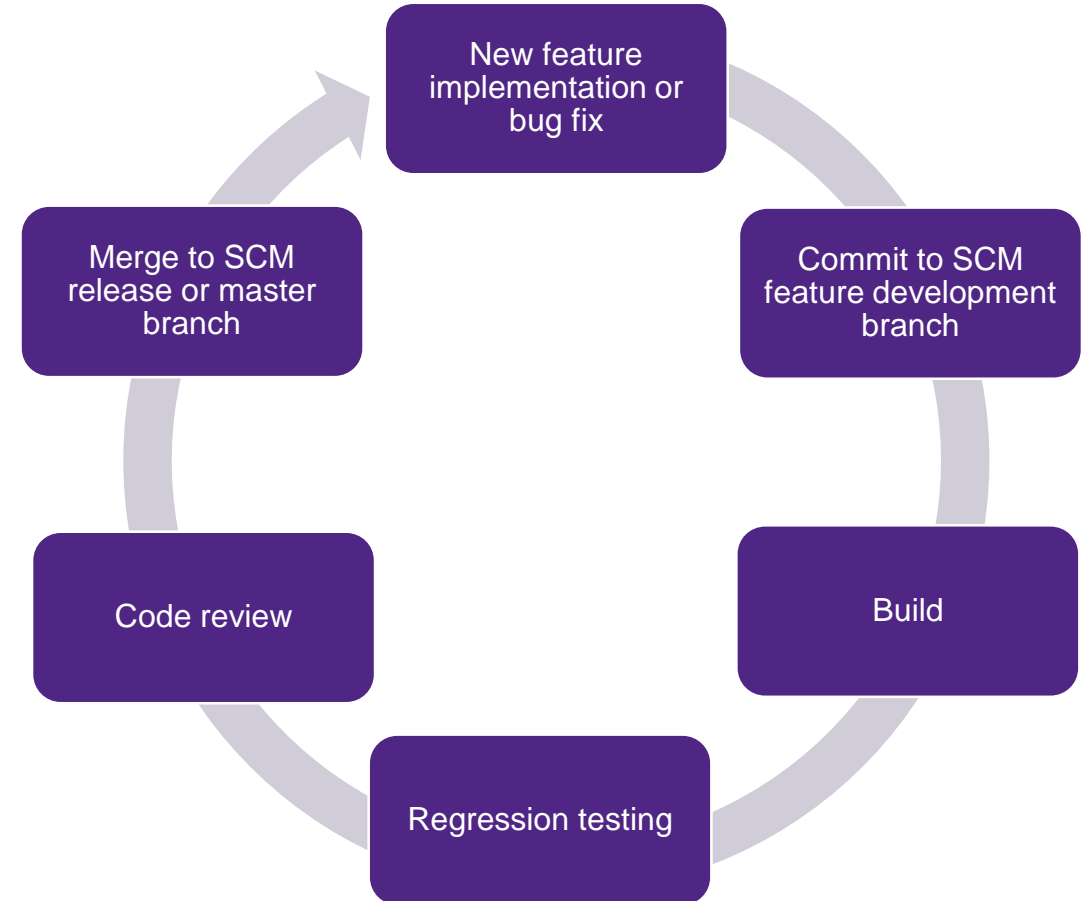
SW and HW for the same SoC can be verified totally separately

Would it be possible to adopt some best practices from SW world?

Would be even possible to combine the verification flows (SW and HW)?

What is Continuous Integration (CI)?

- Software development practice requiring developers to check-in source code changes regularly, upto several times a day.
 - Small deviations are easier to debug
 - Each check-in is validated quickly
 - As the code is safe, it can be merged with other check-ins (automatically)
- Jenkins is industry's preferred CI automation tool



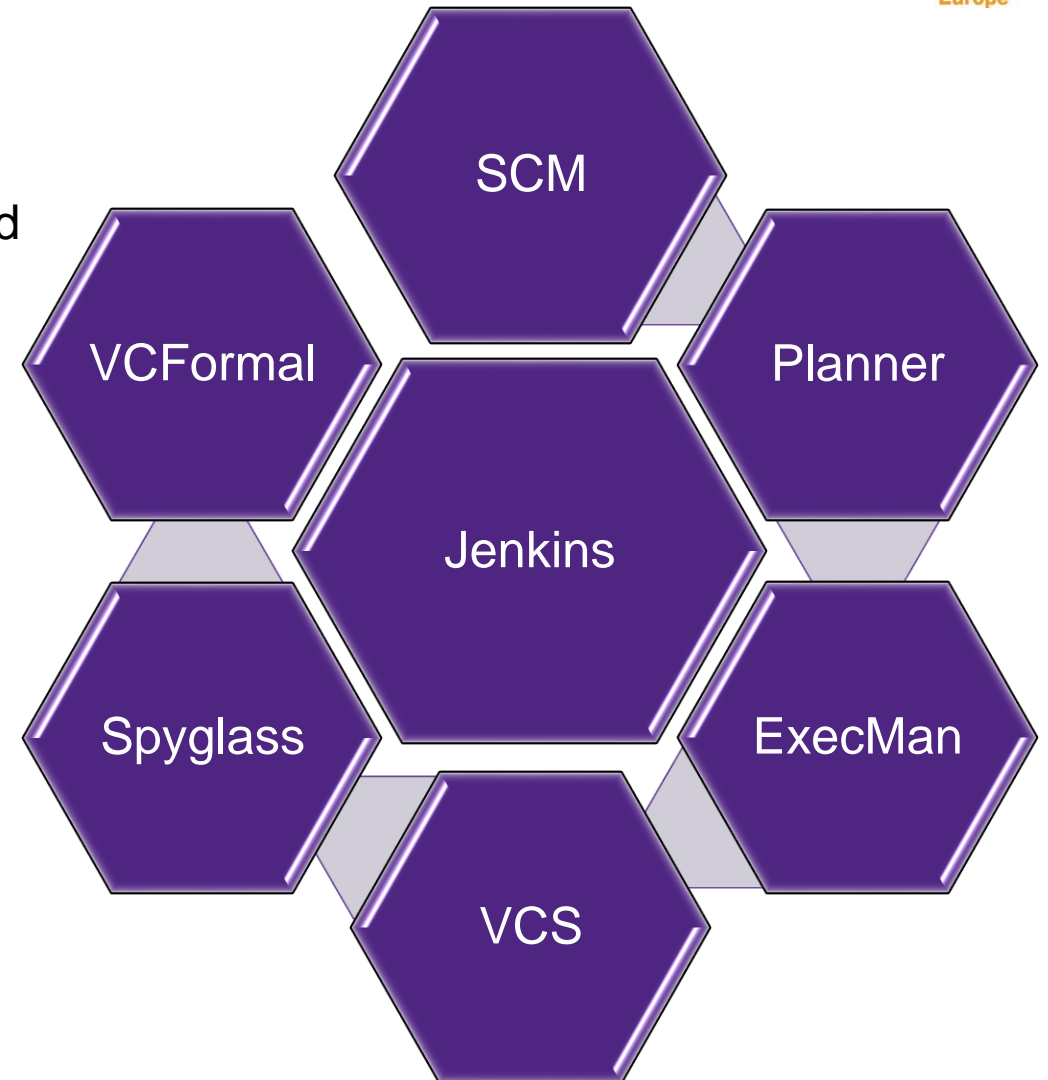
What is Jenkins?

- Automation server used Continuous Integration flow
 - Developed at Sun Microsystems around 2004 (Hudson)
 - Java-based, open source
- Brings automation of CI flow
 - Can trigger builds (and start regressions) automatically if code is checked-in
 - Can also use time-based triggers (cron jobs)
- Runs tools and scripts that are available on the work area
- Usability can be extended by different Plugins
 - Over 1400 plugins available in repository covering many different applications
- Allows different types of job configurations (freestyle, pipeline, multijob)
- Web-based dashboard enabling team and management to get status information
 - Plugins available to represent status of regressions; tables, trend graphs etc.
 - Plugins available for several other collaboration tools (e.g. Jira, Confluence, Git)
- East to set up (run the command `java -jar jenkins.war`)
- Good for SW testing (integrated built tools, short tests, not much data stored)
- More productized commercial enterprise versions available (Jenkins 2.0 by Cloudbees)



Jenkins and Hardware Verification

- Jenkins knows nothing about hardware verification
 - What tools to run and under what conditions
 - Need to provide scripts for Jenkins to run compilation and tests
 - Tool can have own plugins to ease the usage and to visualize results
- By default Jenkins can trigger regressions but cannot interpret results
 - Doesn't understand UVM status messages and pass/fail criteria in regression
 - Doesn't know how to execute coverage merging
 - Cannot show status of regression in real time
- Adding parallelism is possible, but not that easy
- There are tools that does these tasks very well and Jenkins can run those, for example Execman

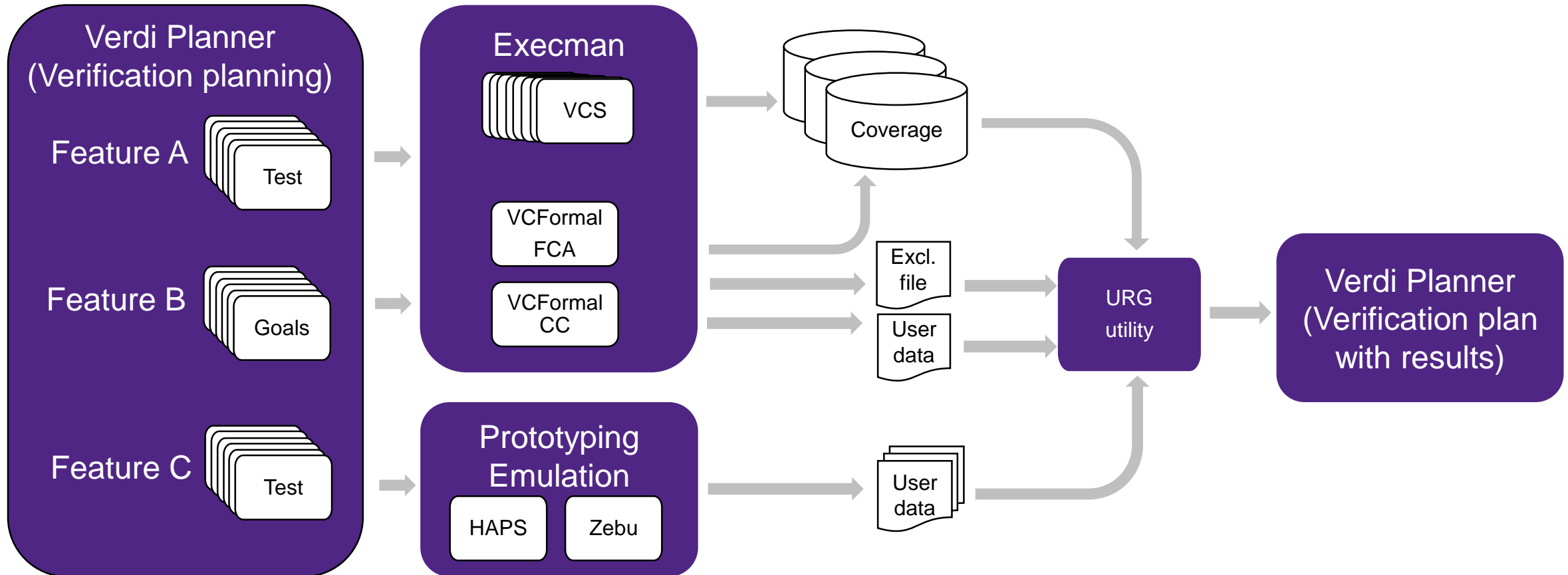


Agenda

- Introduction
- SW development practices
- **Building continuous regression flow**
- Example setups
- Summary

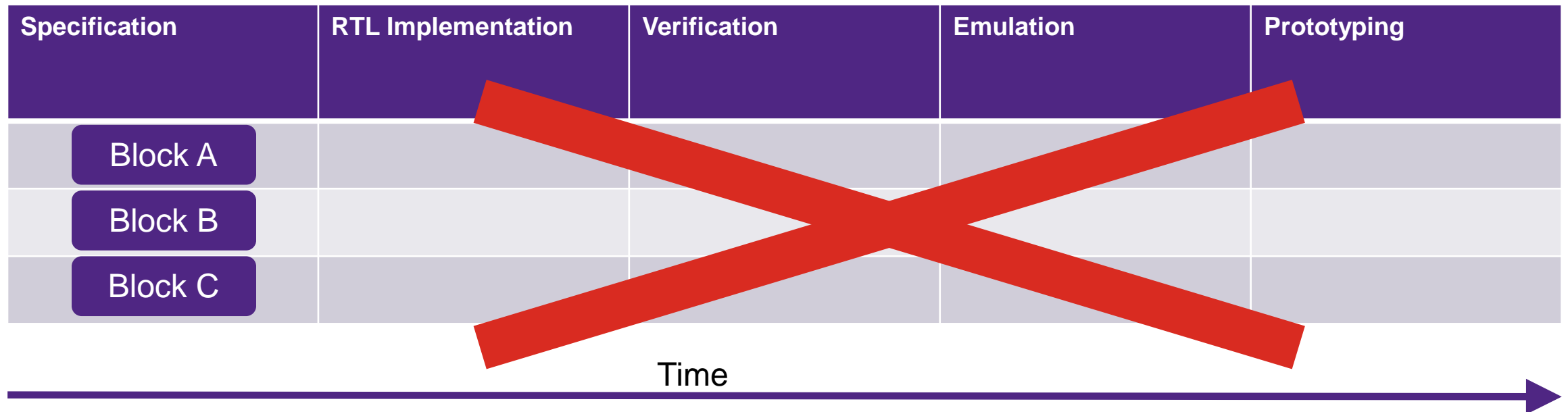
Feature driven verification

Example SoC verification flow setup



Traditional verification flow

RTL implementation centric approach



All the parts of the SoC need to be ready before moving to next step in the project

Feature driven verification flow

Doing the most important things first

Specification	RTL Implementation	Verification	Emulation	Prototyping
Feature A				
Feature B				
Feature C				

Time



Minimum set of design need to be ready before moving to next step in the project

Integrated Coverage Aware Planning with Verdi

Coverage

Verification Plan

Req. Spec

The screenshot displays the Verdi:vdCoverage:1 application interface. The left pane shows a hierarchy of modules with their respective coverage scores and line counts. The middle pane shows a detailed view of the verification plan, including a table of conditions and FSM states. The right pane shows a requirement specification document for the UART16550 core.

Name	Score	Line
test_jukebox	56.87%	83
cd1	92.11%	100
fifo1	76.82%	88
jb1	76.83%	90
st0	55.82%	82
coin1	61.99%	80
kp1	60.66%	89
st1	59.41%	85
coin1	67.40%	83
kp1	60.55%	89
st2	59.21%	85
coin1	66.73%	83
kp1	62.44%	89
st3	52.64%	82
st4	56.10%	82
trkgen[2].c...	0.00%	0
trkgen[3].c...	0.00%	0
trkgen[4].c...	0.00%	0
trkgen[5].c...	0.00%	0
trkgen[6].c...	0.00%	0

Name	Line	Cond	FSM	To
49 F49	C1	89.66%	54.05%	
49.1 F5	C1	89.66%	54.05%	
49.2 F5	C1	89.66%	54.05%	
M50		89.66%	54.05%	
50 F52	C1	85.45%	42.86%	64.71%
50.1 F5	C1	83.95%	40.74%	64.71%
50.2 F5	C1	89.66%	45.95%	
M52		89.66%	45.95%	
51 F55	C1	85.45%	47.25%	58.82%
52 F58	C1	82.73%	45.05%	52.94%
52.1 F5	C1	80.25%	40.74%	52.94%
52.2 F6	C1	89.66%	51.35%	
53 F61	C1	82.73%	46.15%	52.94%
54 F64	C1	88.24%		
55 F65	C1	100.0%		
56 F66	C1	96.00%	42.86%	100.0%
57 G1	C1			

SPEC.1 plan.hvp-UART_spec.pdf

64%

Linked

OpenCores UART16550 core specifications 8/1/2002

Bit #	Access	Description
		cleared upon reading from the register.
		'0' - Otherwise.

48 Modern Status Register (MSR)

The register displays the current state of the modern control lines. Also, four bits also provide an indication in the state of one of the modern status lines. These bits are set to '1' when a change in corresponding line has been detected and they are reset when the register is being read.

Bit #	Access	Description
0	R	Delta Clear To Send (DCTS) indicator
1	R	'1' - The CTS line has changed its state.
2	R	Delta Data Set Ready (DDSR) indicator
3	R	'1' - The DSR line has changed its state.
4	R	Trailing Edge of Ring Indicator (TER) detector. The RI line has changed its state from low to high state.
5	R	Delta Data Carrier Detect (DDCD) indicator
6	R	'1' - The DCD line has changed its state.
7	R	Complement of the CTS input or equals to RTS in loopback mode.
8	R	Complement of the DSR input or equals to DTR in loopback mode.
9	R	Complement of the RI input or equals to Out1 in loopback mode.
10	R	Complement of the DCD input or equals to Out2 in loopback mode.

49 Divisor Latches

The divisor latches can be accessed by setting the 7th bit of LCR to '1'. You should restore this bit to '0' after setting the divisor latches in order to restore access to the other registers that occupy the same addresses. The 2 bytes form one 16-bit register, which is internally accessed as a single number. You should therefore set all 2 bytes of the register to ensure normal operation. The register is set to the default value of 0 on reset, which disables all serial I/O operations in order to ensure explicit setup of the register in the software. The value set should be equal to (system clock speed) / (16 x desired baud rate). The internal counter starts to work when the LSB of DL is written, so when setting the divisor, write the MSB first and the LSB last.

plan.hvp X

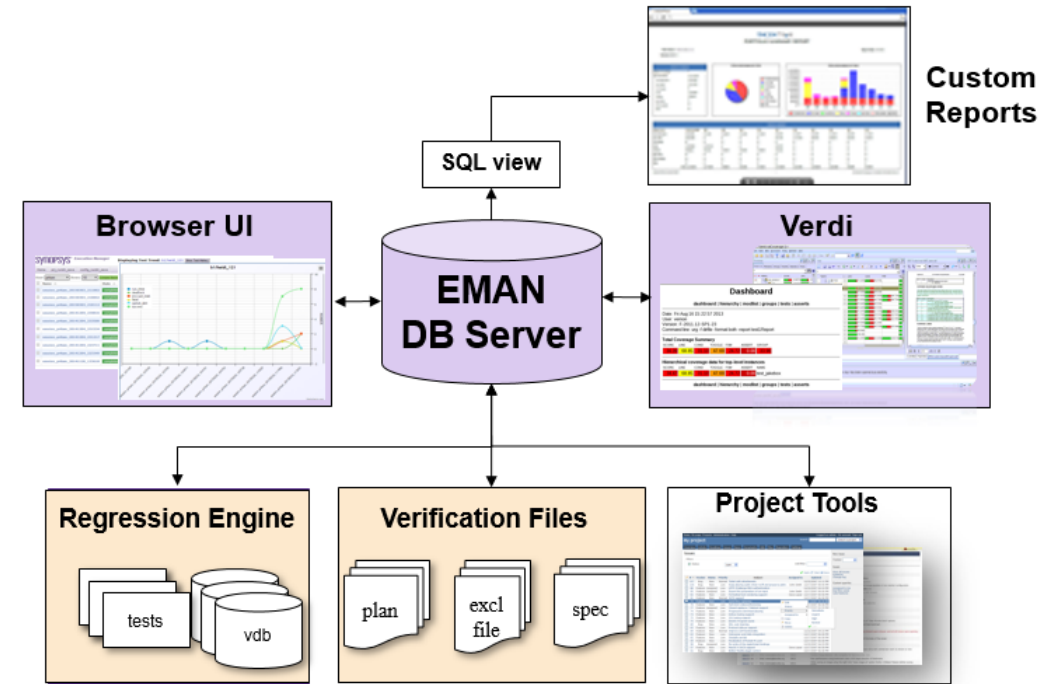
CovSrc.1 Hvp

CovDetail HvpDetail SPEC.1 plan.hvp-UART_spec.pdf

Traceable Verification Across Flow and Tools for Faster Verification Closure

What is Execman?

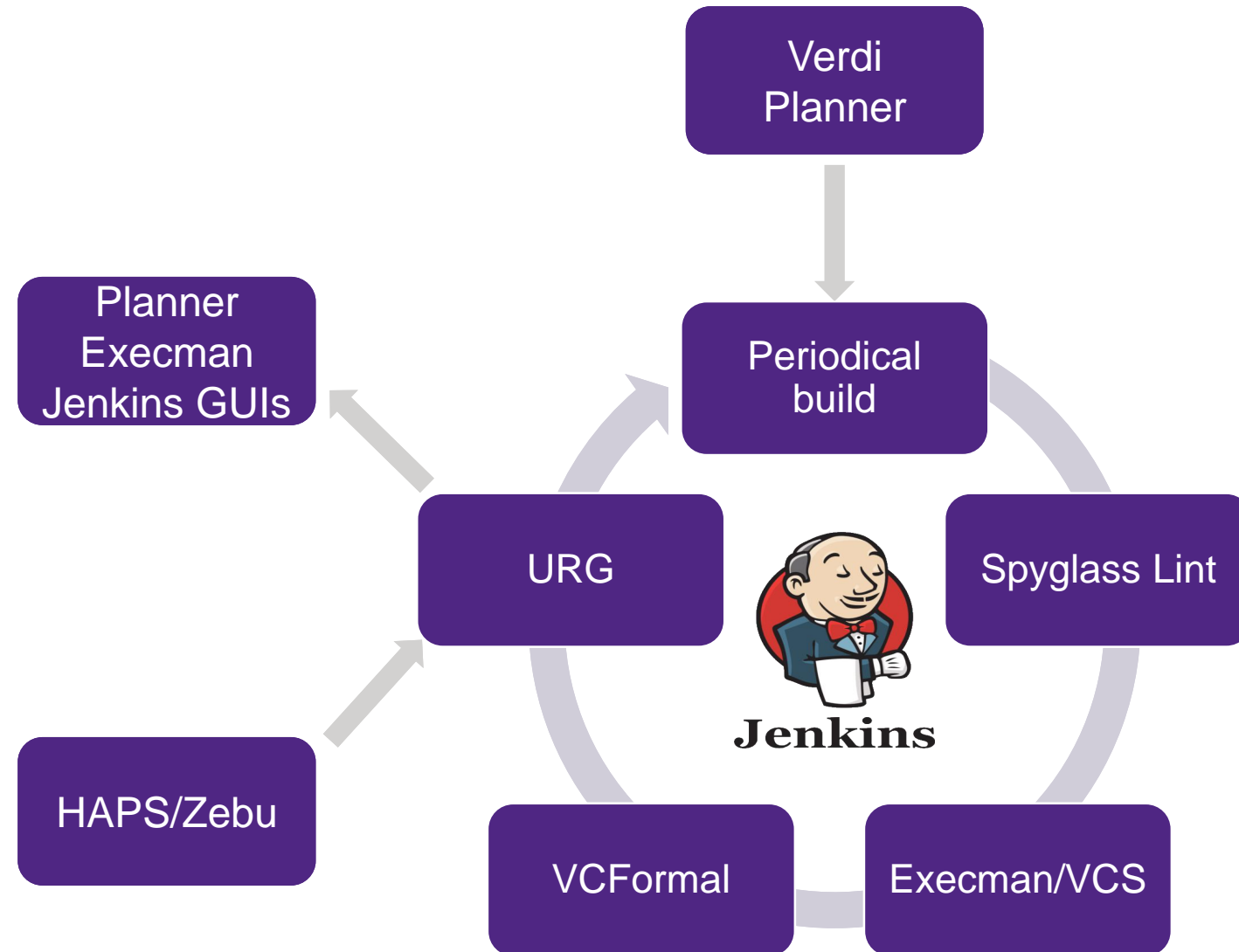
- Server-client solution providing execution management for SoC verification
- Can be executed from terminal shell or Verdi Planner
 - Launches and monitors jobs in server farm
- Deploys SQL database
 - Efficient for collecting lot of data
 - Provides APIs for custom solutions
- Real time monitoring through Verdi or Web server
- Supports verification plan with custom metrics
- Random test order option
- Support for running long test set in batches
- Automatic debug runs
- Multiple builds and build specific test sets
- Configurable log file parser to detect all the bugs
- ...



Continuous regression flow

SoC level example

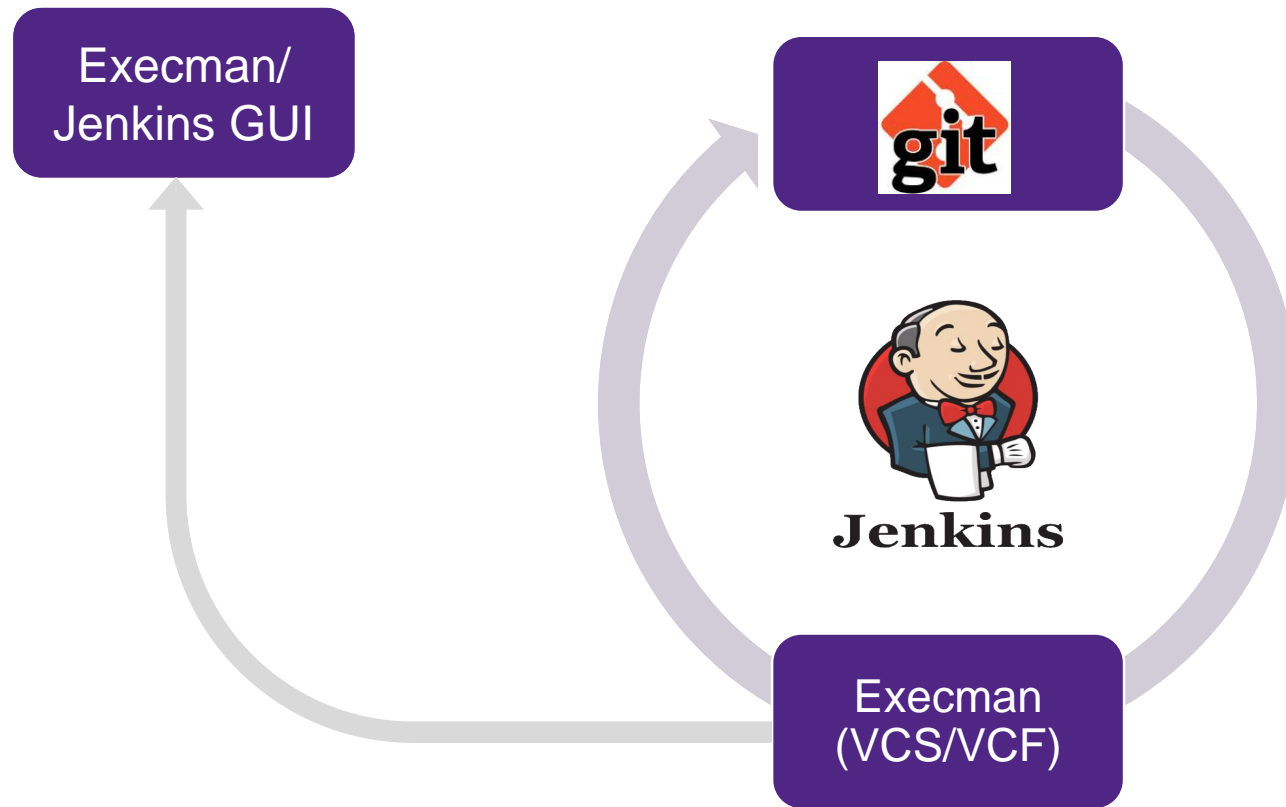
- Jenkins pipeline
- Time triggered Jenkins project (“nightly build”)
- Possible to run full test set in smaller batches



Continuous regression flow

Block level example

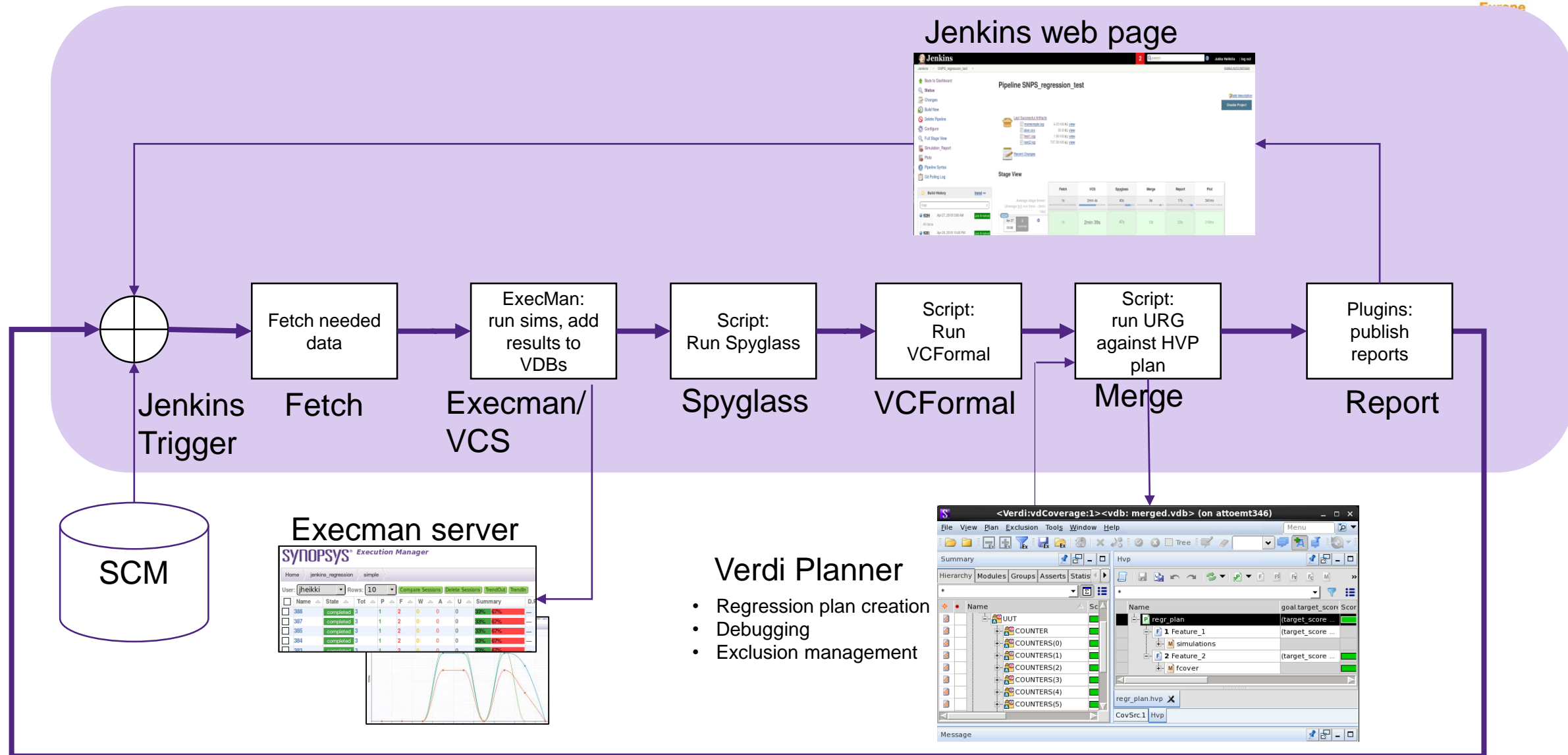
- Jenkins triggered from source code management
- "Smoke set" simulation
- Execman Jenkins Plugin



Agenda

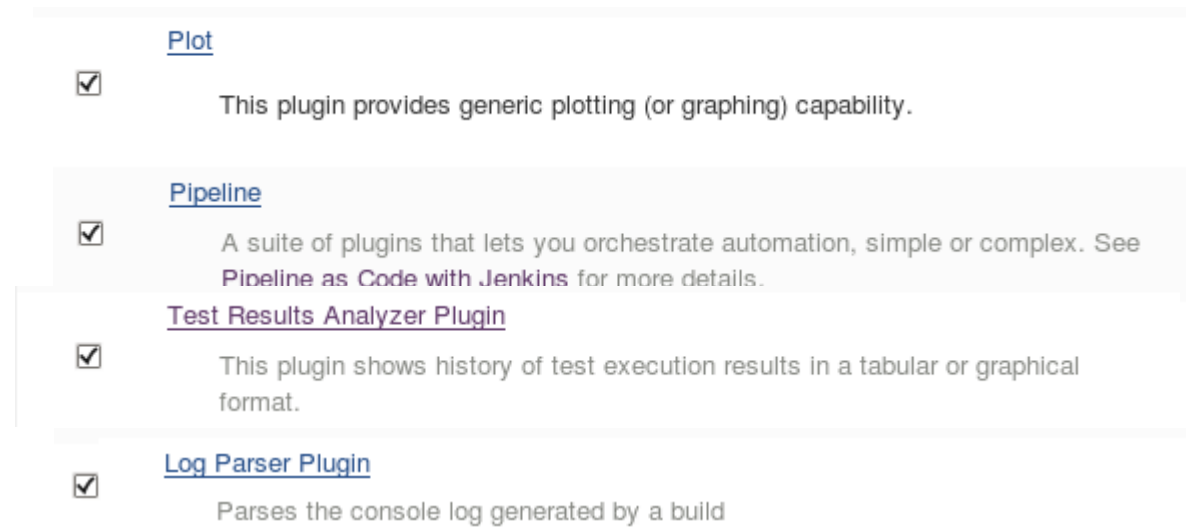
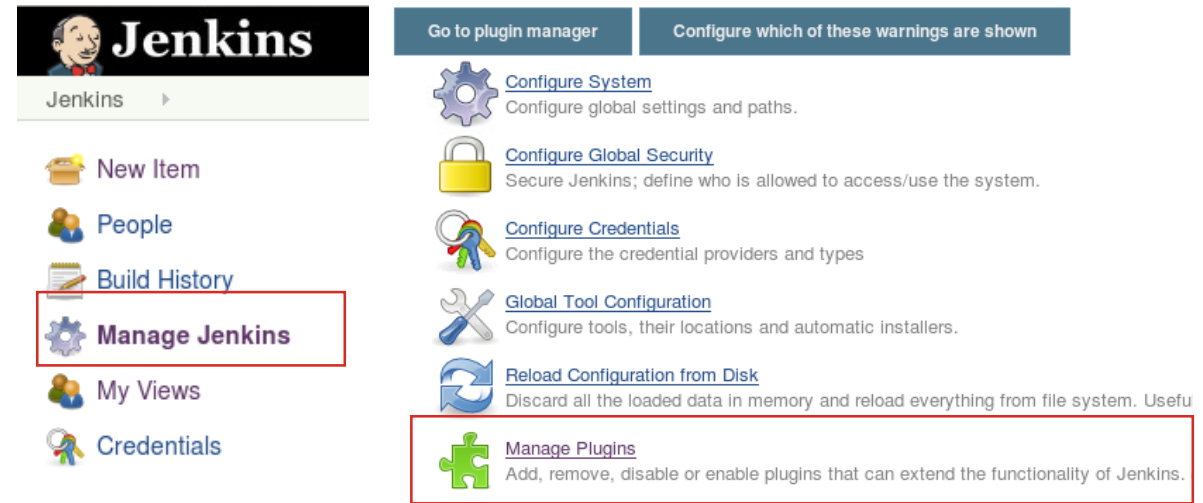
- Introduction
- SW development practices
- Building continuous regression flow
- **Example setups**
- Summary

Jenkins pipeline Example



How to install required plugins

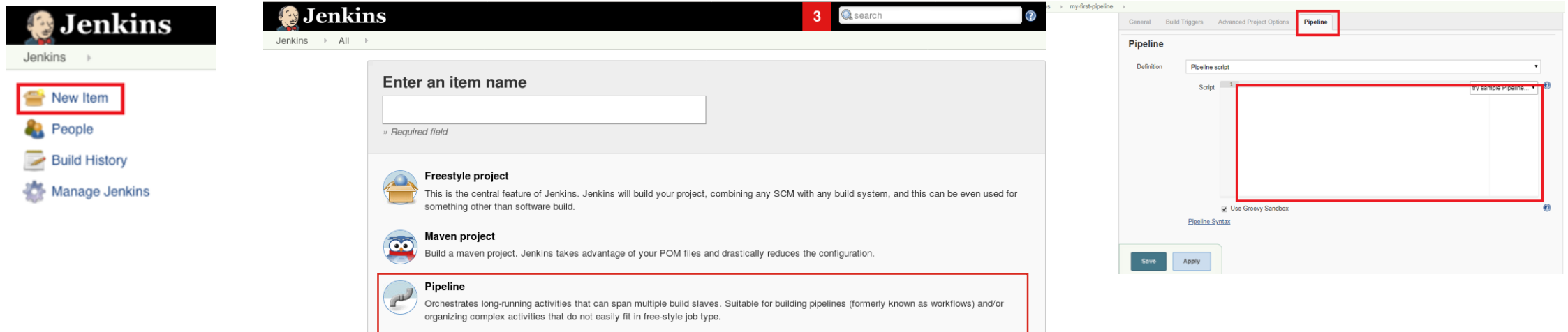
1. Manage Jenkins -> Manage Plugins
2. Search required plugins
3. Select and install



How to implement Jenkins Pipeline

Getting started

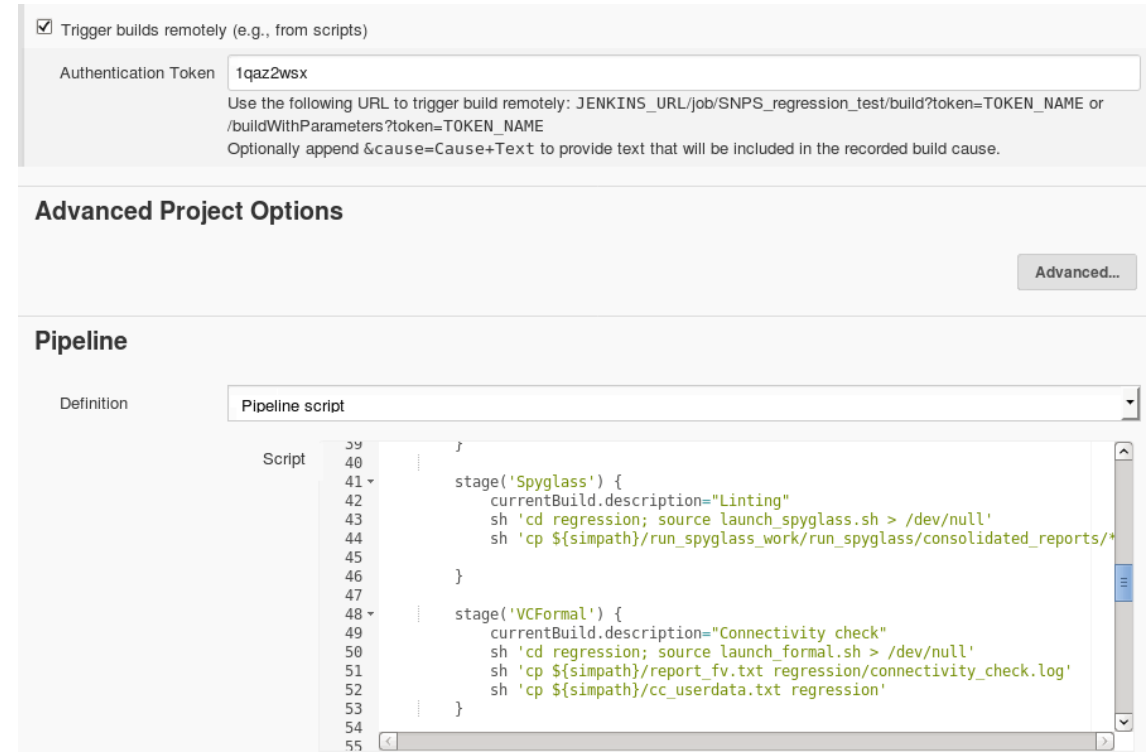
1. First, log on to your Jenkins server and select “New Item” from the left panel:
2. Next, enter a name for your pipeline and select “Pipeline” from the options.
Click “Ok” to proceed to the next step
3. You can now start working your Pipeline script



Run tools in pipeline

Configuration

- Pipeline is described in Jenkins using Groovy syntax
 - Different tools are being executed in different pipeline stages
 - Actual plugins are being called at the end of pipeline to collect results and draw graphs
- Select trigger according to you needs
 - In case of remote trigger with token additional configuration may be needed in version control (e.g. Git hooks) to get commits to launch builds



☒ Trigger builds remotely (e.g., from scripts)

Authentication Token

Use the following URL to trigger build remotely: JENKINS_URL/job/SNPS_regression_test/build?token=TOKEN_NAME or /buildWithParameters?token=TOKEN_NAME
Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.

Advanced Project Options

Pipeline

Definition

Script

```
39 | }
40 |
41 | stage('Spyglass') {
42 |     currentBuild.description="Linting"
43 |     sh 'cd regression; source launch_spyglass.sh > /dev/null'
44 |     sh 'cp ${simpath}/run_spyglass_work/run_spyglass/consolidated_reports/*
45 |
46 | }
47 |
48 | stage('VCFormal') {
49 |     currentBuild.description="Connectivity check"
50 |     sh 'cd regression; source launch_formal.sh > /dev/null'
51 |     sh 'cp ${simpath}/report_fv.txt regression/connectivity_check.log'
52 |     sh 'cp ${simpath}/cc_userdata.txt regression'
53 | }
54 |
55 | }
```

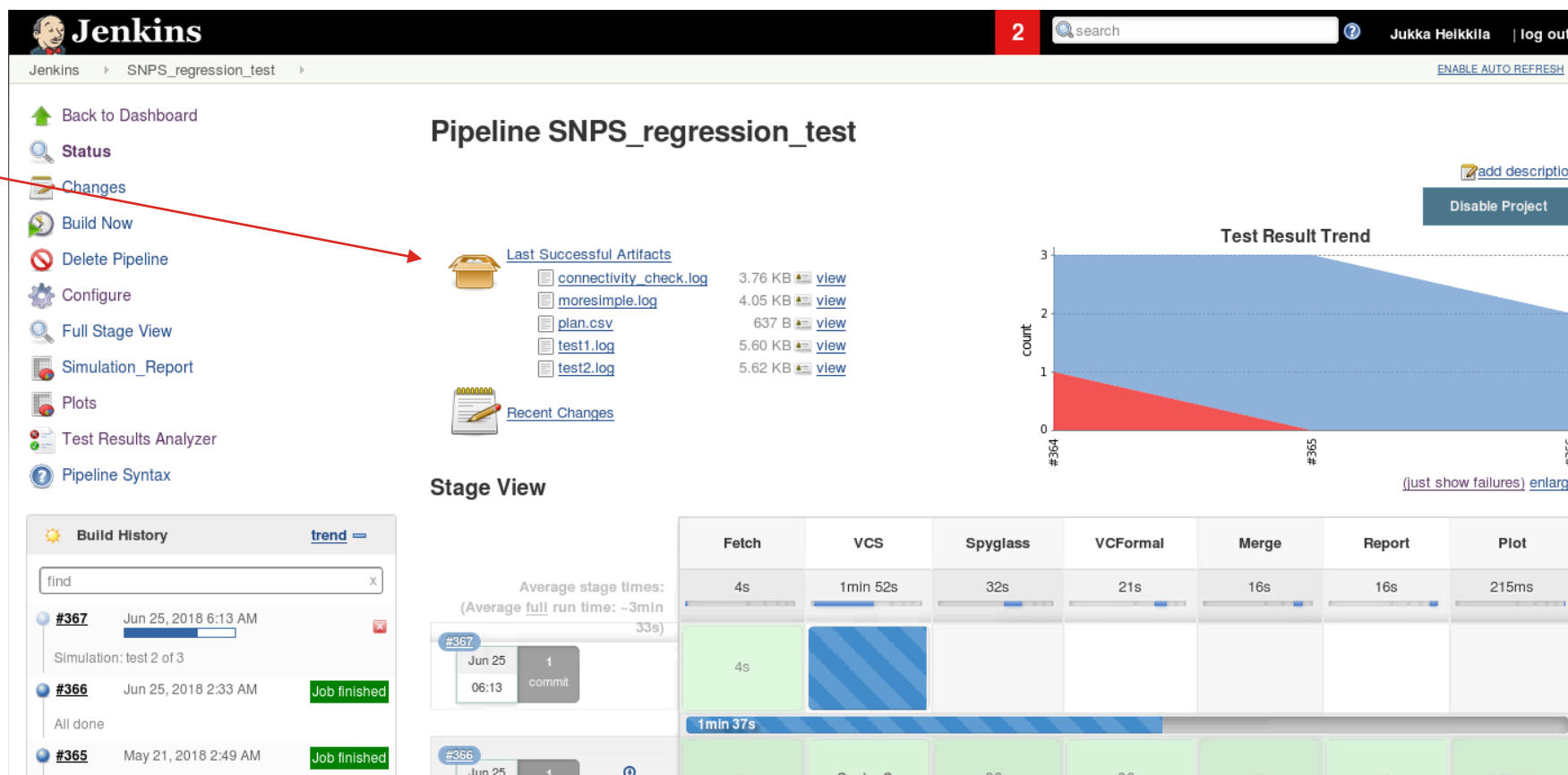
Visualizing the job results (Pipeline project)

Jenkins project page

Collected artifacts

More graphs from plugins

The results of last runs

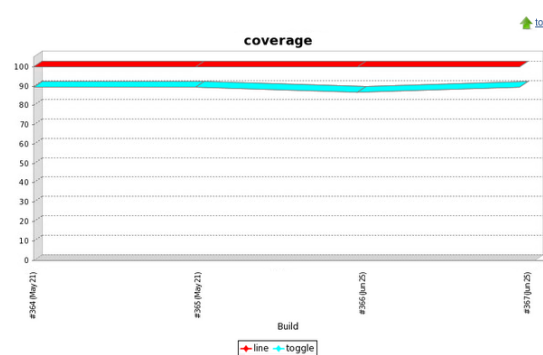
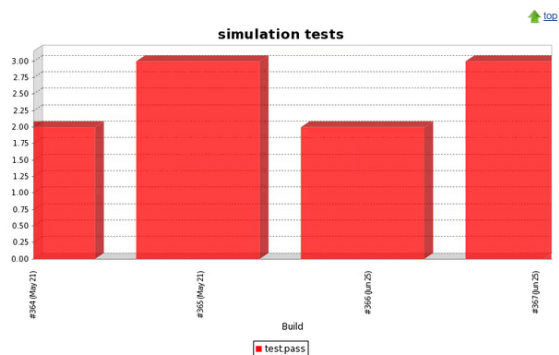
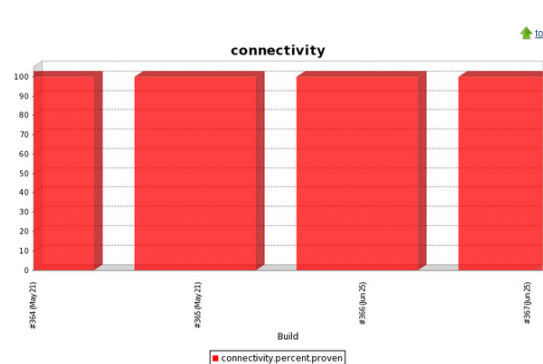
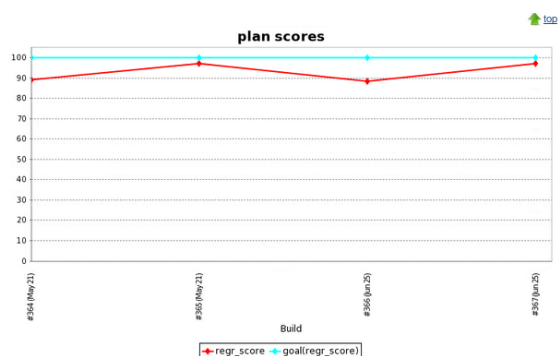


Provided by
common
plugins

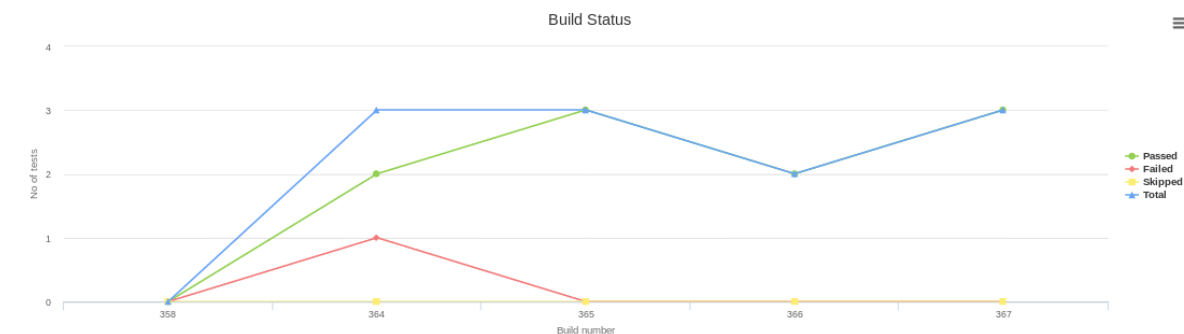
Pipeline stages

Visualizing the job results (Pipeline project)

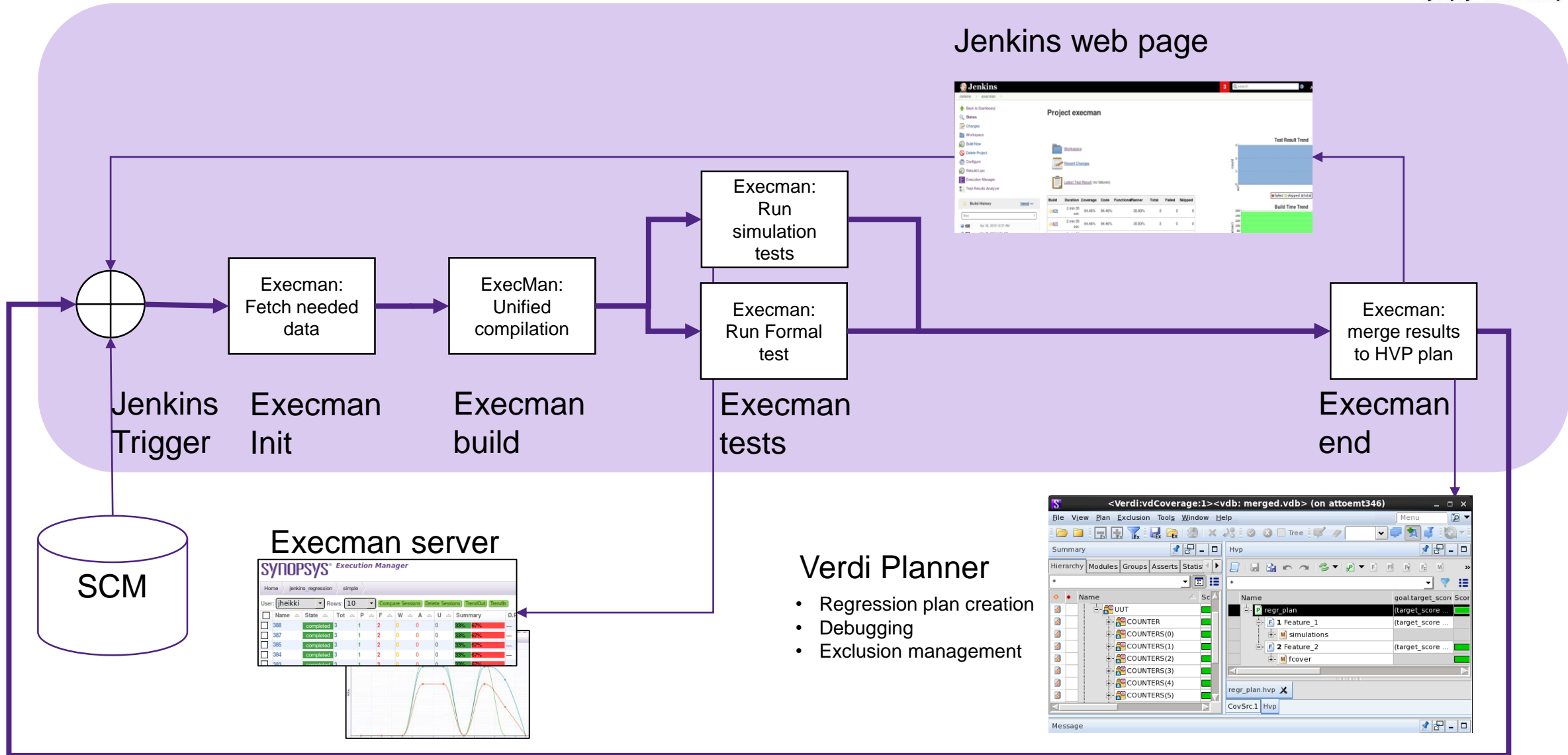
Graphs from "Plot" and "Test result analyzer" plugins



New Failures	Chart	See children	Build Number ⇒ Package-Class-Testmethod names ↓	367	366	365	364	358
	✓	●	(root)	PASSED	PASSED	PASSED	FAILED	N/A
	✓	●	test1	PASSED	PASSED	PASSED	PASSED	N/A
	✓	●	test2	PASSED	PASSED	PASSED	PASSED	N/A
	✓	●	test3	PASSED	N/A	PASSED	FAILED	N/A

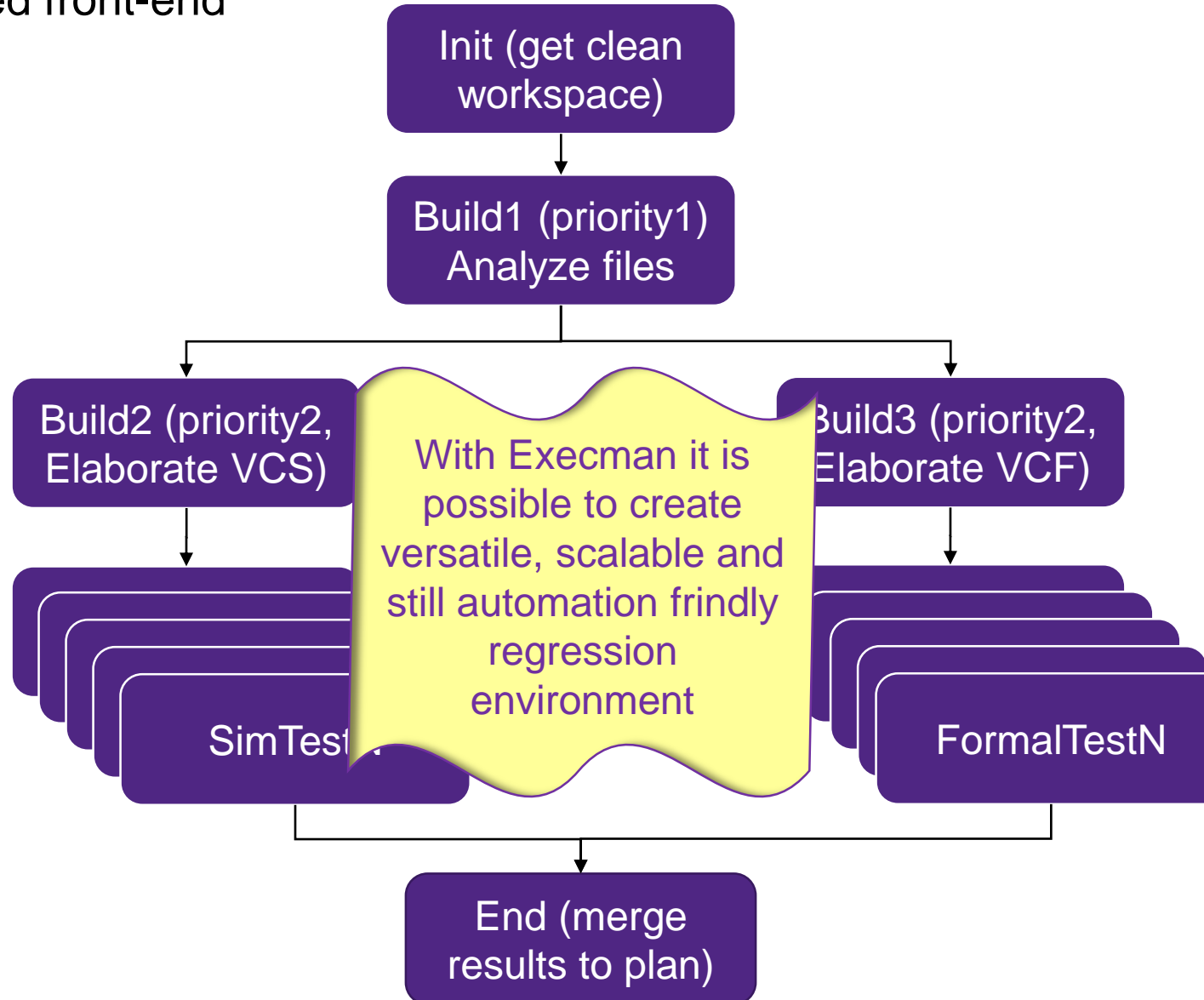


Jenkins Plugin Example



Running multiple tools in parallel with Execman

With unified front-end



init:

- run_cmd: cd \$regr_dir; git clone ...

builds:

- name: Build1
run_dir: \$regr_dir/build_common
run_cmd: make clean analysis
priority: 1

- name: Build2
run_dir: \$regr_dir/build_vcs
run_cmd: make elab_vcs
priority: 2

- name: Build3
run_dir: \$regr_dir/build_vcf
run_cmd: make elab_vcf
priority: 2

tests:

- name: SimTest1
run_cmd: make test_sim UVM_TEST= ...
build: Build2

- name: FormalTest1
run_cmd: make test_vcf MODE=CC
build: Build3

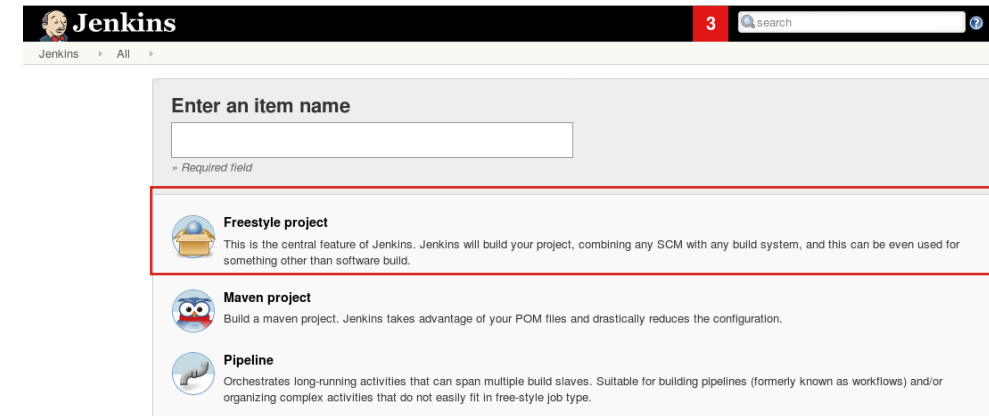
...

end:

- run_cmd: cd \$regr_dir/merge; urg

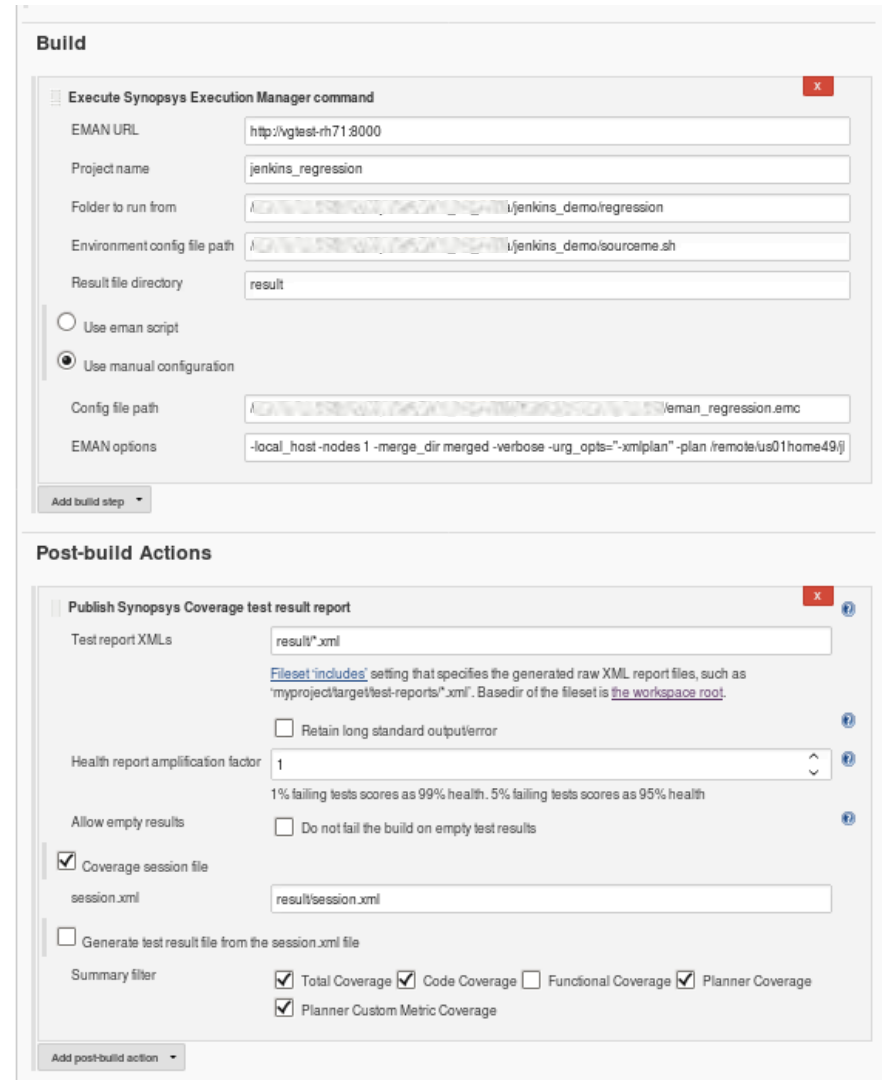
How to use Jenkins Execman Plugin

1. First, log on to your Jenkins server and select “New Item” from the left panel:
2. Next, enter a name for your pipeline and select “Freestyle project” from the options. Click “Ok” to proceed to the next step
3. Install the Plugin like described earlier
4. You can now start working your build configuration



Configure the Plugin

- Add build step for Synopsys Execution Manager
 - Configure server and project details
- Add Post-build Action for publishing test results
 - Select metrics you want to see in Jenkins



The screenshot shows the Jenkins configuration page for the Synopsys Execution Manager plugin. It is divided into two main sections: 'Build' and 'Post-build Actions'.

Build Section:

- Execute Synopsys Execution Manager command:** This section contains several input fields:
 - EMAN URL:** `http://vgtest-rh71.8000`
 - Project name:** `jenkins_regression`
 - Folder to run from:** `/jenkins_demo/regression`
 - Environment config file path:** `/jenkins_demo/source.sh`
 - Result file directory:** `result`
- Configuration Options:** There are two radio buttons:
 - ☐ Use eman script
 - ☒ Use manual configuration
- Config file path:** `/eman_regression.amc`
- EMAN options:** `-local_host-nodes 1 -merge_dir merged -verbose -urg_opts="-xmlplan" -plan /remote/us01/home49/j`

Post-build Actions Section:

- Publish Synopsys Coverage test result report:** This section contains several configuration options:
 - Test report XMLs:** `result*.xml`
 - Fileset 'includes' setting:** A text area with the following content: `Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is the workspace root.`
 - Retain long standard output/error:** ☐
 - Health report amplification factor:** `1`
 - Allow empty results:** ☐ Do not fail the build on empty test results
 - Coverage session file:** ☒ `session.xml` `result/session.xml`
 - Generate test result file from the session.xml file:** ☐
 - Summary filter:** ☒ Total Coverage ☒ Code Coverage ☐ Functional Coverage ☒ Planner Coverage ☒ Planner Custom Metric Coverage

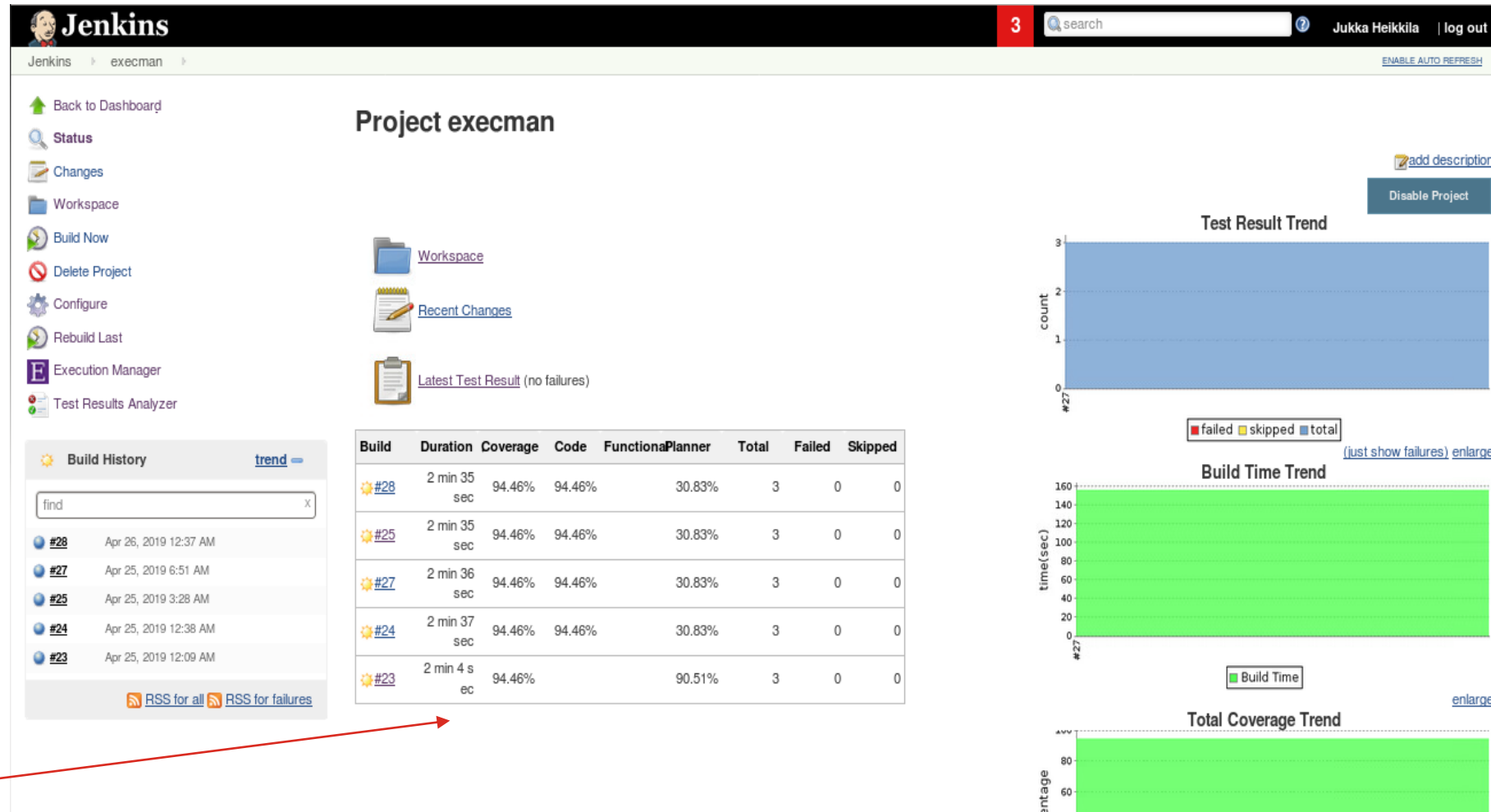
Visualizing the job results (Execman plugin project)

Jenkins project page

More graphs from
Execman
and plugins

Previous
Builds

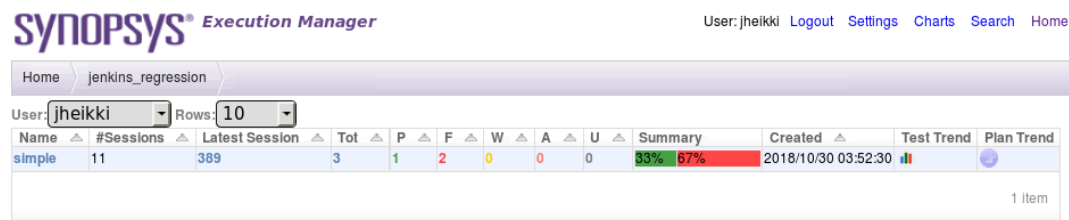
Test
results



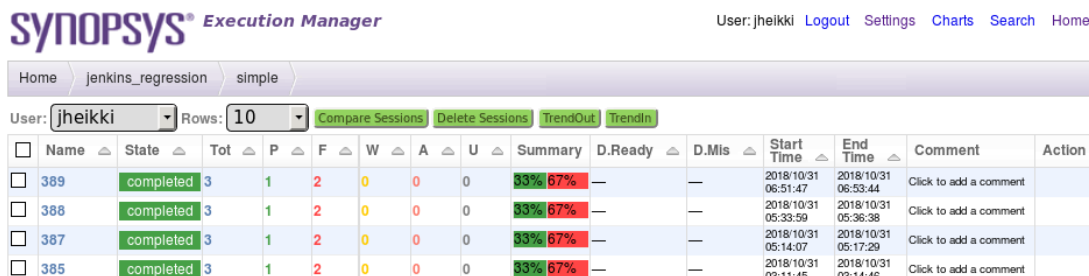
Visualizing the job results (both examples)

Execman web GUI

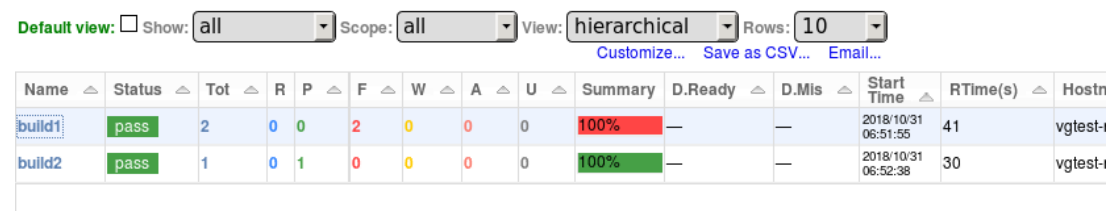
Project
page



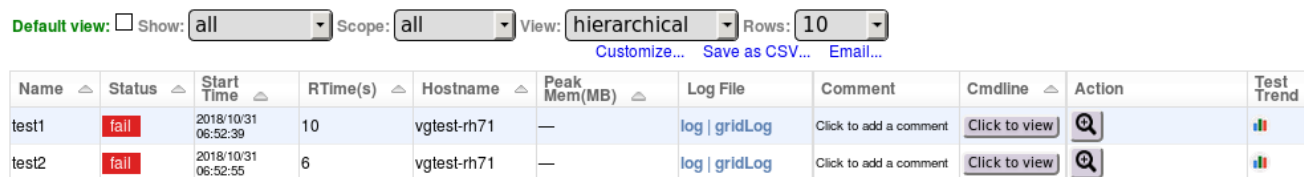
Config
page



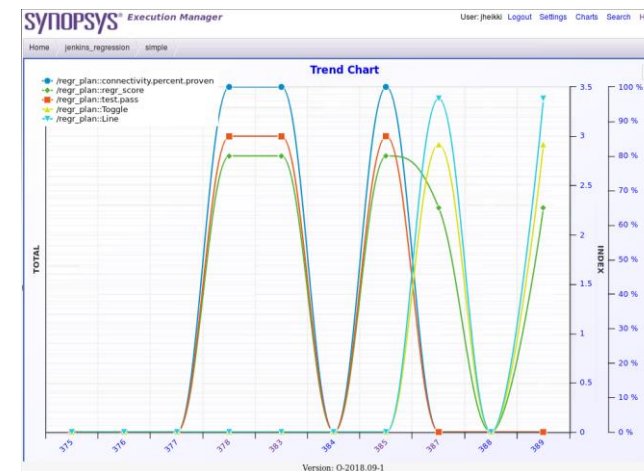
Build
page



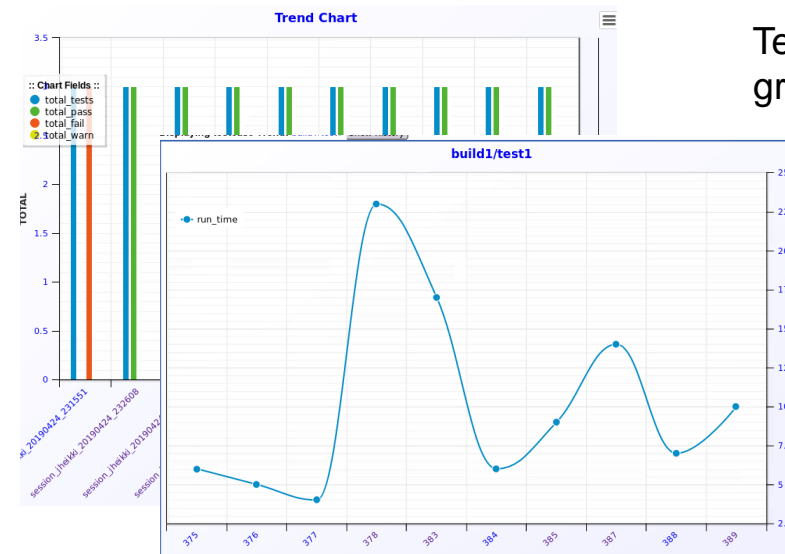
Test
page



Plan
graphs



Test
graphs



Visualizing the job results (both examples)

urgReport HTML report

- Can be published in Jenkins as such using *HTML publisher* plugin

[Back to SNPS regression test](#) [dashboard](#) [Zi](#)

dashboard | hierarchy | modlist | groups | tests | asserts | userdata | hvp | trend

HVP Hierarchy regr_plan Feature_1

Feature : regr_plan.Feature_1

SCORE	LINE	TOGGLE	connectivity	proven	falsified	connectivity.percent.proven	regr_score	regr_score_icm2	regr_score_icm3	test	pass	fail	warn	assert	unknown	test.percent.pass
94.70	100.00	89.39	10	10	0	100.00	97.35	97.35		2	2	0	0	0	0	100.00

Measure Totals:

SCORE	LINE	TOGGLE	connectivity	proven	falsified	connectivity.percent.proven	regr_score	regr_score_icm2	regr_score_icm3	test	pass	fail	warn	assert	unknown	test.percent.pass
94.70	100.00	89.39	10	10	0	100.00	97.35	97.35		2	2	0	0	0	0	100.00

Measure simulations:
Metrics: test, test.percent.pass, regr_score, regr_score_icm2
Goal: (regr_score >= 100) (regr_score_icm2 == 90)

SCORE	LINE	TOGGLE	connectivity	proven	falsified	connectivity.percent.proven	regr_score	regr_score_icm2	regr_score_icm3	test	pass	fail	warn	assert	unknown	test.percent.pass
							100.00	100.00		2	2	0	0	0	0	100.00

Sources:

test2

NAME	SCORE	LINE	TOGGLE	connectivity	proven	falsified	connectivity.percent.proven	regr_score	regr_score_icm2	regr_score_icm3	test	pass	fail	warn	assert	unknown	test.percent.pass
test2								100.00	100.00		1	1	0	0	0	0	100.00

test3

NAME	SCORE	LINE	TOGGLE	connectivity	proven	falsified	connectivity.percent.proven	regr_score	regr_score_icm2	regr_score_icm3	test	pass	fail	warn	assert	unknown	test.percent.pass
test3								100.00	100.00		1	1	0	0	0	0	100.00

Measure connectivity:
Metrics: Line, Toggle, Assert, connectivity, connectivity.percent.proven, regr_score, regr_score_icm2
Goal: (connectivity.proven == 1) (regr_score >= 100) (regr_score_icm2 == 90)

SCORE	LINE	TOGGLE	connectivity	proven	falsified	connectivity.percent.proven	regr_score	regr_score_icm2	regr_score_icm3	test	pass	fail	warn	assert	unknown	test.percent.pass
			10	10	0	100.00	100.00	100.00								

Sources:

CON1

NAME	SCORE	LINE	TOGGLE	connectivity	proven	falsified	connectivity.percent.proven	regr_score	regr_score_icm2	regr_score_icm3	test	pass	fail	warn	assert	unknown	test.percent.pass
CON1				1	1	0	100.00	100.00	100.00								

Visualizing the job results (both examples)

Verdi Planner

- Detailed results can be viewed in Verdi Planner

<Verdi:vdCoverage:1><vdb: merged.vdb> (on attoemt366)

File View Plan Exclusion Tools Window Help

Summary

chy Modules Groups Asserts Statistics Tests

Data from 4 tests was loaded.

Test Name/Test Location	Type	Status
merged/test	Test	
test1--(test1/test...)	Test Record	pass
test2--(test2/test...)	Test Record	pass
test3--(test3/test...)	Test Record	pass

Category Value

Hvp

Name	regr_score	Line	test.pass	connectivity	connectivity.prov	connectivity.prov	conr
regr_plan	99.72%	99.72%	3	8	8	100.0%	
1 Feature_1	100.0%	100.0%	1	8	8	100.0%	
simulations			1				
connectivity				8	8	100.0%	
coverage		100.0%					
2 Feature_2	99.52%	99.52%	2				
fcover	66.67%	66.67%	1				
simulations			1				
coverage		100.0%					

Agenda

- Introduction
- SW development practices
- Building continuous regression flow
- Example setups
- **Summary**

Facing the challenges

How to plan the verification efficiently?

Feature driven verification with Planner enables project to reach the goal step by step

What is the overall verification status?
Information can come from multiple sources

Using Planner/Execman flow enables progress tracking with data from several sources

How can this information be made visible to users/management?

Execman SQL server with web interface provides scalable and flexible way of distributing information

Verification time gets long as designs grow

Distributed parallel simulations with multicore option is the best solution for this

Utilizing R&D infra efficiently for regression simulations (grid, multicore, configs)

Execman is the tool for this. Excluding this part of the flow to Execman makes the regression flow simpler to maintain

Launching functional and static checks in FE development takes lots of time for small changes

Some CI platform, like Jenkins, can be harnessed here to bring automation

Building and testing regression solutions for Jenkins is difficult

You can build and test environment with Execman and when ready, just add "eman" command in Jenkins

How to keep SW and HW in sync?

Using Synopsys Virtual prototyping tools it is possible to run co-simulations with real SW

Summary

- SoC verification is complex subject and automation is needed
- Jenkins is a widely used general purpose automation platform, but it is not capable of running large scale SoC verification flow as such
- It is possible to implement parts of CI flow in SoC development, or start using existing CI infrastructure
- Verdi Planner enables smart verification planning with custom metrics for comprehensive progress monitoring
- VCS/VCFormal/Spyglass provides the leading engines for workstation verification
- ExecMan is a final piece of a puzzle providing computing farm control, efficient SQL based database for data storage and real-time progress monitoring



Thank you!

