www.phparch.com

November 2016
VOLUME 15 - Issue 11

Moving Forward

php[architect]

ALSO INSIDE

Education Station:
Do You Truly Know Your Tool of Choice?

Community Corner:
Getting the Most out of a Conference

Leveling Up:
Perpetual Process ~~Improvement~~ Perfectionization

Security Corner:
To Scan or Not to Scan

finally{}:
Can We Be Nice to Each Other?

Strangler Pattern, Part Two: Beginning to Design for Scale With RabbitMQ

What Are Interfaces, Abstracts, and Traits?

Creating PHP Extensions With Zephir

Dev Divas: History's Heroines of Computing, Part Two

Atlanta
May 2017

photo by tableatny: https://www.flickr.com/photos/53370644@N06/

php[tek] 2017

Call for Papers opening soon!

Subscribe to our mailing list, or follow us on
Twitter & Facebook for announcements

tek.phparch.com

php[architect]

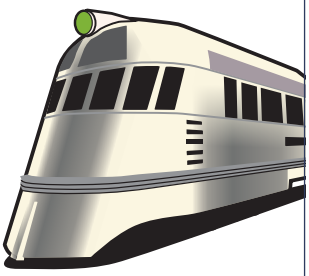
CONTENTS

NOVEMBER 2016
Volume 15 - Issue 11

Moving Forward

3
Strangler Pattern, Part Two: Beginning to Design for Scale With RabbitMQ
Edward Barnard

11
What Are Interfaces, Abstracts, and Traits?
Chilion Snoek



17
Creating PHP Extensions With Zephir
Victor Bolshov

26
Dev Divas: History's Heroines of Computing, Part Two
Vesna Vuynovich Kovach

Columns

2 Editorial:
Moving Forward
Oscar Merida

33 leveling up:
Perpetual Process ~~Improvement~~ Perfectionization
David Stockton

37 Community Corner:
Getting the Most out of a Conference
Cal Evans

40 Education Station:
Do You Truly Know Your Tool of Choice?
Matthew Setter

47 Security Corner:
To Scan or Not to Scan
Chris Cornutt

49 October Happenings

51 finally{}:
Can We Be Nice to Each Other?
Eli White

Subscriptions
Print, digital, and corporate subscriptions are available. Visit https://www.phparch.com/magazine to subscribe or email contact@phparch.com for more information.

Advertising
To learn about advertising and receive the full prospectus, contact us at ads@phparch.com today!

Managing Partners
Kevin Bruce, Oscar Merida, Sandy Smith

php[**architect**] is published twelve times a year by:
musketeers.me, LLC
201 Adams Avenue
Alexandria, VA 22301, USA

Contact Information:
General mailbox: contact@phparch.com
Editorial: editors@phparch.com

Print ISSN 1709-7169
Digital ISSN 2375-3544

Moving Forward

It's only natural to hesitate when you're starting down a new road. Sure, it helps to be prepared, but the first couple of steps can be daunting. For me, it helps to know others have done this before—or at least something similar. This month, we look at how different people have managed to move forward to learn and build new things.

> *"Destroying the status quo because the status is not quo"*
>
> *–Dr. Horrible*

Impostor syndrome is rampant in our industry. I know I've felt like a bit of a fraud. To be honest, that's probably why I overstayed at my first job even though the work had gotten repetitive and made me a bit too cynical. If the only enjoyment you're getting is predicting how the next project will fail, its time to try something new. When the chance to work at D.C. United—for new readers, I should add that I'm a huge soccer and United fan—I hesitated. Despite having built and launched numerous websites over the preceding eight years, I still was afraid that I couldn't possibly put one together. In the end, with the encouragement of my friends and family, I took the job. I learned so much on that job, not just technically, but about my own capabilities, skills, and limits which have helped me to this day. Plus I got to play futbol on the field at RFK stadium.

Don't let yourself stand still.

First of all, in this issue we have, *Dev Divas: History's Heroines of Computing, Part Two* concludes Vesna Kovach's exploration of the pivotal contributions made by women to programming. I won't give anything away here, but I was surprised to just how many of the tools and techniques we use today were made by women. They had to essentially invent programming to work on ENIAC!

Victor Bolshov will show you how to create your own PHP extensions in *Creating PHP Extensions With Zephir*. If you have operations or tasks that are performance bottlenecks, Zephir can speed it up. Zephir transpiles PHP-like syntax into C so you don't have to know as much about the internal workings of the Zend engine to create an extension.

Are you still wondering *What Are Interfaces, Abstracts, and Traits?* Chilion Snoek will explain using them to make your code more reusable and easier to understand. If you work with a modern framework—or a CMS like Drupal—you'll find Interfaces, Abstract Classes, and Traits under the hood.

Edward Barnard continues his series in *Strangler Pattern, Part Two: Beginning to Design for Scale With RabbitMQ*. This time, he describes the microservices which add new features to a legacy application. See how he used PHP, RabbitMQ, and Redis to scale his application's capacity.

In *Education Station: Do You Truly Know Your Tool of Choice?*, Matthew Setter looks at popular text editors and IDES to look at how to use them effectively when writing code. Even if already have a favorite tool, you might learn a thing or two by checking out the competition.

Are you still doing things the same way because "that's always how we've done them?" Then you need to read David Stockton's *Leveling Up*. He describes how they've improved their bug tracking and deployment processes at work in *Perpetual Process ~~Improvement~~ Perfectionization*. You're sure to find some advice you can begin to adapt and follow in your own work. Don't accept the status quo if it's making your work a big pain point.

Chris Cornutt explains what kind of security scanners are available for PHP in *Security Corner: To Scan or Not to Scan*. He covers the different kinds of scanners available to look for security issues in your code base and looks at the Free and Commercial options that are out there.

If you're going—or plan to go—to your first conference soon, don't skip Cal Evans' *Community Corner*. He's rounded up what he's learned as a conference regular for you in *Getting the Most out of a Conference*. Don't dive into your next conference without this guide.

And no matter where you're interacting with your peers and colleagues, Eli White has one rule to follow in *finally{}: Can We Be Nice to Each Other?*.

What Did You Think?

Do you have a question about an article? Want to share feedback about this issue? Let us know! Leave a comment on the issue page, reach us via twitter @phparch, or ask our contributors via twitter or email.

Write For Us

If you would like to contribute, contact us, and one of our editors will be happy to help you hone your idea and turn it into a beautiful article for our magazine.

Visit https://phpa.me/write or contact our editorial team at write@phparch.com and get started!

Stay in Touch

Don't miss out on conference, book, and special announcments. Make sure you're connected with us via email, twitter, and facebook.

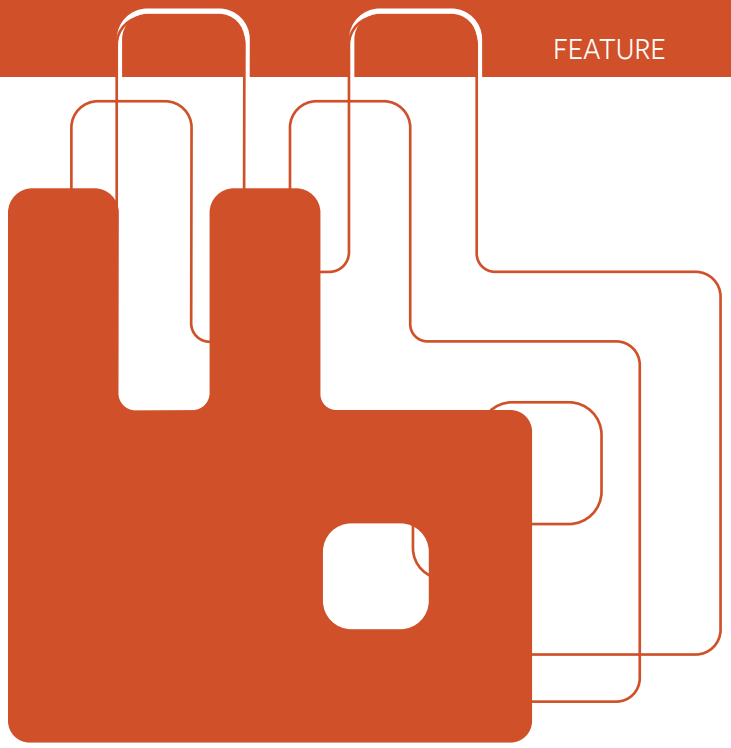
- Subscribe to our list: http://phpa.me/sub-to-updates
- Twitter: @phparch
- Facebook: http://facebook.com/phparch

Download this Issue's Code Package:

http://phpa.me/November2016_code

Strangler Pattern, Part Two: Beginning to Design for Scale With RabbitMQ

Edward Barnard



We scaled data collection at InboxDollars.com from 50K views to 600K views a day by offloading much of our work to PHP/MySQL microservices. We created a distributed processing system based on CakePHP 3, Redis, and RabbitMQ. *Part Two* shows our microservices infrastructure; we ensured our DevOps team had visibility into the running services and problems that arose. We built resiliency into our system based on experience. We kept things as simple as we could manage.

By time our system went into production, our member views had already increased to 600K views per day. We made a second iteration allowing for at least 4X more traffic capacity.

The Strangler Pattern, Part One laid out our design approach:

- Identify parts of the *new* feature work which can be offloaded from our web servers.
- Implement the offloaded portions using the latest PHP and CakePHP available.
- Integrate these offloaded portions in such a way that we can evolve with newer PHP and CakePHP versions.

We're building a microservices architecture, but we are *not* rewriting our large web application as microservices. We're focused *only* on identifying work to be offloaded, and implementing that work in our newer software environment.

Our web application, naturally, has many features and portions of features which could be offloaded to our Strangler Pattern system. Since there is no direct company-revenue gain in moving stuff around which already works, we carefully evaluate such projects for *other* gain.

Pain Points

We know our "pain points" from experience. We have some MySQL tables "too big to fail." That is, altering the table to add a column, an index, etc., would knock down the site for several hours. That's not acceptable.

For example, we have some very old tables in danger of overflowing their `INTEGER` primary key (4 billion rows). The oldest rows have been archived-off, so the table itself is manageable. We could alter those tables to change the primary key column's size from `INTEGER` (4 bytes wide) to `BIGINT` (8 bytes wide).

Unfortunately, it's not so simple. Legacy stuff rarely is! We have those "too big to fail" tables with foreign-key references to those primary keys. The MySQL `alter table` command, in this specific case, locks the complete table. These tables are central to *all* primary site traffic. That's how the tables got "too big to fail!" Since each `alter table` command takes several *hours* to complete, we'd need to take the entire site offline for the better part of the day to alter those foreign-key columns from `INTEGER` to `BIGINT`. We don't want to do that.

Strangler Pattern to the Rescue

Our *Stranger Pattern* infrastructure provides us a way to solve this problem for several years to come. Given we have a direct revenue benefit from *not* taking down our main site for several hours, we've justified our *Strangler Pattern* project during its first month of production.

Revenue and Earnings

InboxDollars[1] members can, among other things, earn cash for various activities. A member might complete an offer or a survey. The advertising or market-research company pays us, and we, in turn, pay the member a portion of that money. We have a *revenue* table recording what those companies should be paying us, and an *earnings* table recording the money awarded to the member. Both tables are like a transaction log; that is, we insert a new row for each revenue/earnings event.

1 *InboxDollars:* http://www.inboxdollars.com

Table Velocity

We have a new feature, our *TV Channel* wherein members get paid for watching short videos preceded by advertising videos. This changes the velocity (number of table rows inserted per second per table) of our Revenue and Earnings tables. Before this feature, a member might complete an offer with a certain dollar amount. Any given member might trigger a few revenue/earnings table rows per day.

With the TV Channel, we award cash a penny at a time. Someone earning $2.00 per day watching our TV Channel would be triggering 200 revenue/earnings inserts per day. In other words, we are suddenly inserting *many* more rows per day. We did the math, and we need to seriously worry about those 32-bit primary keys!

Pending Credits

To solve this, we created *pending credits*. We tally the current day's credits (for this feature) in a "pending credits" table. Members can open their "recent earnings" page and see those credits immediately. The pending credits become real credit the following day.

This means all of the member's TV Channel credits become a single revenue/earnings table insert. This slower velocity keeps the "too big to fail" tables safe for several more years (at current projections).

Our `tv_pending_credits` table looks approximately like this:

- `Id` (big int): primary key
- `Archive_month` (tiny int): We partition and purge (archive-off) by calendar month
- `Member_id` (int): The member being credited
- `Credit_queue_id`: Nonzero once credit applied
- `Pending_revenue`: Updated each revenue/earnings event
- `Pending_earnings`: Ditto
- `Pending_date`: We have one row per member per day

Fire and Forget

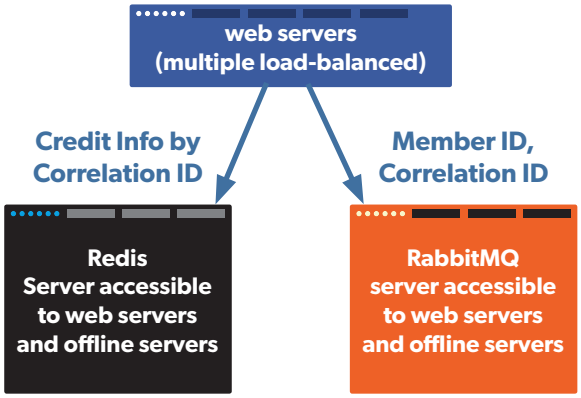
Once our web server has determined this member should be awarded cash credit for the TV Channel view, the web server sends a message into the offline system. This is a "fire and forget" operation meaning the web server sends the message to RabbitMQ and does not wait for a response.

Figure 1 shows the actual processing flow:

1. Generate a unique token as the *Correlation Identifier*[2] (Correlation ID).
2. Place the crediting information in Redis indexed by that Correlation ID.
3. Build a message for RabbitMQ with the member ID and Correlation ID.
4. Send the message to RabbitMQ with *exchange* `bats_realtime` and *routing key* `tv_events`.

2 *Enterprise Integration Patterns page 163, see the paragraph which follows.*

Figure 1. Fire and Forget



My choice to use *both* Redis and RabbitMQ appears unusual. You may recognize this as the *Claim Check (346)* from *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.*[3] The intent of *Claim Check* (the 346 is the page number of the hardcover book) is to reduce the data volume of messages sent through the system. Indeed, RabbitMQ performs best with short messages, so there is value here.

However, in my integration testing, I was able to "break" a RabbitMQ queue in such a way no worker was able to consume any more messages from that queue, see *RabbitMQ unable to get heartbeat working with php-amqplib*[4]. My only recourse was to delete the queue, losing all messages in that queue. We have not seen this problem in production; however, it does point out that distributed systems provide you with *many* new ways for things to fail.

Our Redis-then-RabbitMQ procedure helps ensure we won't lose crediting requests. The crediting information is saved by Correlation ID, with a timestamp as part of the crediting information, in Redis as part of the `tv_events` hash. Both our Redis and RabbitMQ configurations persist their data to disk, but we can and do treat Redis as very fast in-memory cache.

Once per hour, our offline worker "sweeps" through the `tv_events` hash looking for any crediting information placed in Redis more than a few minutes ago. We assume any such event found has been "orphaned" and process it immediately.

We have a single worker which processes pending credits and does the sweep. When it's doing the sweep it can't be processing credits, and therefore we have no timing or race conditions to worry about when doing the sweep.

Redis appears not to be atomic when adding things to a

3 *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.:* https://www.amazon.com/dp/0321200683/

4 *RabbitMQ unable to get heartbeat working with php-amqplib:* http://stackoverflow.com/questions/39006691/

hash. When sweeping through the `tv_events` hash with a Redis `HashIterator`, I found some of the crediting-information values to be empty. I assume this is due to several web servers adding values to the hash at the same time as the single worker is sweeping. Thus, we filter sweep results down to non-empty values with a relatively old timestamp.

Remember, the *normal* process is to receive that Correlation ID via a RabbitMQ message. It would be silly to process values in the sweep we are about to process via the incoming RabbitMQ message. Upon processing the RabbitMQ message, we'd be dealing with crediting information which has already been processed by the sweep. That's an ugly race condition.

Our offline worker, or any part of the offline system, could "tip over." This could happen overnight on a weekend. We want to recover everything intact once things restart.

What does this mean for our worker?

At worker startup, we could have tens or even hundreds of thousands of messages awaiting processing. Our worker has capacity of 50 messages per second. Our worker would, therefore, need quite some time to catch up.

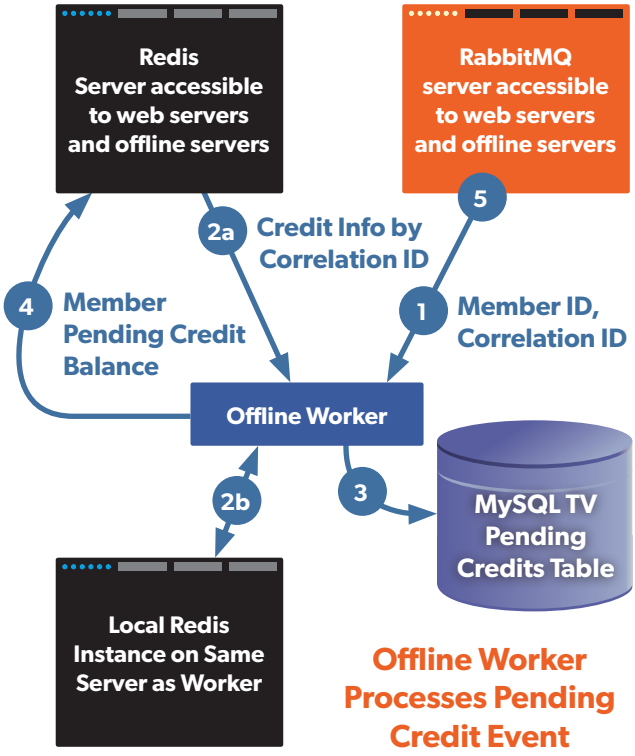
Meanwhile, our worker would have dropped into its sweep cycle and proceeded with "out of band" processing of those "orphaned" messages, not realizing they're all referenced by pending RabbitMQ messages.

We sweep once per hour, but we wait until 20 minutes *after* worker startup. That gives time for 60,000 messages to clear, which hopefully is more than enough. Meanwhile, our DevOps team gets notified any time our RabbitMQ queue hits a backlog of more than a couple thousand messages.

Offline Worker

Figure 2 shows the Pending Credit processing flow:

Figure 2. Process Pending Credit



1. The TV Events offline worker receives a message from RabbitMQ containing the Member ID and Correlation ID (yellow step **1** in the figure).
2. Given the correlation ID, the worker moves the credit information from the Redis Server (step **2a**) to its local Redis instance (step **2b**). By moving the message, it will not be seen in any future sweep for missing messages.
3. The worker adds the pending credit to the TV Pending Credits table with an `INSERT INTO tv_pending credits ... ON DUPLICATE KEY UPDATE ...` statement (step **3**).
4. The worker reads, updates, and writes the member's Pending Credits balance on the Redis Server (step **4**).
5. The worker acknowledges the RabbitMQ message, upon which RabbitMQ deletes the message (step **5**).

Note that the web servers do not have access to the TV Pending Credits table. We've offloaded the database table in addition to the event processing. Instead, we use a *data pump* to keep the web servers up to date. All of the web servers have access to the Redis Server and know to pull Pending Credits from the `pending_credits` hash indexed by Member ID.

We have multiple types of pending credits. The worker is only updating one specific type. The list of pending credits is

stored as a string, a JSON-encoded array. We read the pending credits from Redis (if any), set the TV Channel pending credits, and write the JSON-encoded result back to the Redis Server by member ID.

Given that a member should not be receiving two different types of pending credits simultaneously, there is little concern for two updates (to the same member) stepping on each other. Even if the display value did somehow get trashed, the *actual* pending credits are safe in the MySQL tables.

Data Pump

Sam Newman describes Data Pumps in *Building Microservices*[5] chapter 5:

> *Rather than having the reporting system pull the data, we could instead have the data pushed to the reporting system… the data pump should be built and managed by the same team that manages the service.*

In our case, we have a double win. First, we have moved the MySQL activity off of our web servers and the main database, freeing up our web server and database resources. Second, memory-based retrieval from Redis cache is far faster than a MySQL table read.

Local Redis

We are going to a lot of work bouncing between RabbitMQ, the Redis Server, and the Local Redis instance. The crediting information is small. We could have simply sent everything via RabbitMQ and never used Redis.

Distributed processing brings complexity. We, of course, want to avoid as much extra complexity as possible.

- Much of our offline processing involves collecting and logging data for later use. We prefer to use the MySQL bulk-insert statement wherever practicable. However, the worker only sees one message at a time. We, therefore, need to accumulate the bulk-insert values somewhere. We accumulate them as items in a Local Redis hash.
- We have offline scoring algorithms which hit Redis *hard*. This means a *lot* of traffic to the Redis server. This traffic means network congestion which affects everything else including our member-facing websites.
- The Local Redis instance, by contrast, does not require a network connection. It's a loopback connection on the same server. This means the offline worker can use the Local Redis instance as much as needed without any concern over network traffic impact.
- We can't reliably iterate through a Redis hash when several other servers are adding items to that hash. Instead, we move the item to Local Redis in a hash completely under the worker's control.

5 *Building Microservices:* https://www.amazon.com/dp/1491950358/

The Redis Server, RabbitMQ, Local Redis triad is working well for our initial use cases. Since it works well for us, it makes sense to standardize this structure for *all* of our use cases. Any developer who has learned the pattern can follow that pattern from feature to feature.

Building Trust

Most companies, naturally, focus on projects aimed at increasing company revenue. When the norm is 1–3 week projects or sprints, it's quite rare to take an entire month out to build infrastructure.

It's quite a vote of confidence, in other words, to be given the entire summer of 2016 to build out new infrastructure. Our business projections clearly require the additional capacity; however, doing things differently carries risk. Our company revenue depends on our web site. We can't afford to fail in production.

The flip side of the coin, so to speak, is building trust in the new system. *The Strangler Pattern, Part One* described our whiteboard sessions educating our full development team.

In discussing concerns with our DevOps team, they proposed the idea of local Redis instances for each back-end machine in addition to the globally accessible (to us, not to the public) Redis server. This was based on DevOps' observation of network traffic congestion. This has proven to be a great idea, and is part of the design we've seen here.

I realized, as a point of trust, it was crucial everything have as much visibility as possible to DevOps right out of the gate. I wasn't going to gain trust if they couldn't tell when things were okay or not. Our discussions and reviews became part of the design.

BATS

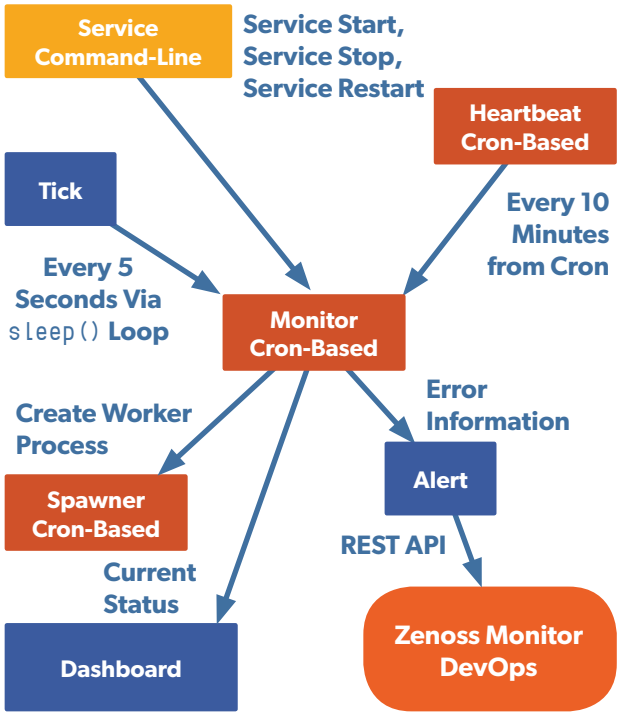
I decided our Strangler Pattern infrastructure should have a name and a logo "to make it a real product." I used the logo as the first slide of my first presentation and everyone accepted it! Our Strangler Pattern system is *BATS*, standing for Batch System.

Figure 3. BATS Logo

Command and Control

Figure 4 shows the BATS components:

Figure 4. BATS Components



BATS Infrastructure Lines Show RabbitMQ Messages Sent

At InboxDollars, we call each command-line process part of BATS a *worker*. That's to distinguish them from other cron jobs which are not part of the BATS distributed-message-processing system. We'll use the same term here.

The `service` task instructs `monitor` to start, stop, or restart a given worker. Each worker is identified as a "service," which is just a convention within the BATS code. However, given developers and system administrators understand the concept of starting and stopping services, the concept plays well.

The `tick` task sends a "tick" message to `monitor` once every five seconds. Various processes need a way to wake up and do work even when they are idle. For example, a worker might be accumulating 1,000 records for a MySQL bulk insert, but things go idle 600 messages in. By seeing the "tick" message, the worker can go ahead and do the bulk insert after enough time has elapsed.

Workers doing a bulk insert do a bulk insert of *all* pending items and immediately delete *only* those Local Redis items which were just inserted. This way, if the worker fails, the next worker will pick up all pending items when it does the next bulk insert.

The `heartbeat` task is fired by the system cron daemon once every ten minutes. It sends a "heartbeat" message to `monitor` and awaits a reply. If it receives no response within a few seconds, or receives a response saying there is a problem, it sends an email to the system administrators. Any time we are rolling code to production and restarting the monitor, we expect to see a "heartbeat" warning while waiting for the monitor to start up again.

The `monitor` task is also fired by cron once every ten minutes. The monitor:

- Sends a "please register" message to all workers every five seconds, triggered by the tick message.
- Keeps track of all running workers based on their incoming "register" messages.
- Starts all non-cron-based workers according to the BATS configuration.
- Prunes stale worker registrations off of its list.
- Restarts "missing" workers.
- Instructs specific workers to stop in response to the `service stop` and `service restart` commands.
- Starts workers in response to the `service start` and `service restart` commands.

To start a worker, `monitor` constructs the command line invocation and sends it as a message to `spawner`. Spawner creates the command-line process. This separation allows monitor to be running on one server and spawner to be running elsewhere.

The `spawner` task is fired by "cron." It has one process running on each server which is part of the BATS system. When monitor needs to start a process, it can send the request to any spawner, or it can send it to a specific spawner, depending on the message routing key.

The `alert` task accepts messages from any worker. It chooses whether or not to send a specific message to our Zenoss monitoring system, and if so, notifies Zenoss via its REST interface.

The `dashboard` task provides visibility into the running BATS system. For the moment, it updates a Redis string with the current list of items running. We can expand it later to visualize anything we need to.

BATS Component Startup

Upon startup, each BATS process generates a unique instance token, so different copies of the same task can be distinguished from each other.

Each of the BATS infrastructure tasks ensures only one copy of itself is running. The BATS system is designed to be spread across multiple servers, though we're only using a single server at present. When starting up, the BATS component sends out two "startup" messages routed to itself, with the message containing its own unique instance token.

If the starting-up component receives an "already running" response, that response came from an active copy of the component. The starting-up component immediately stops.

If the starting-up component does *not* quickly receive a

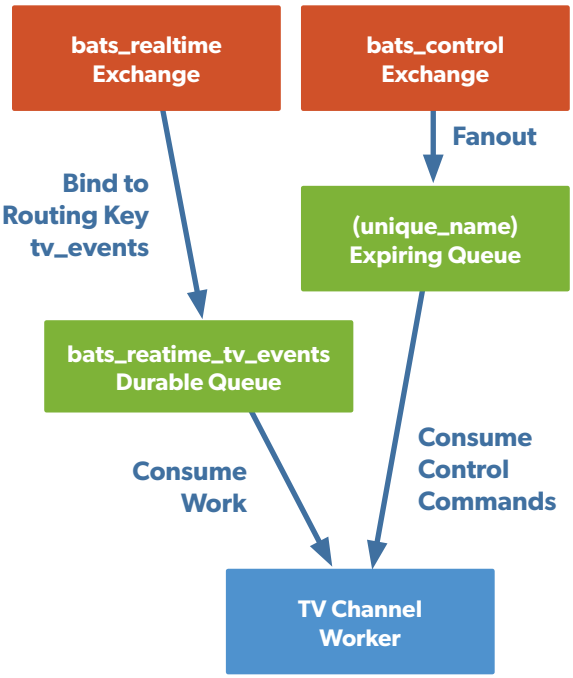
response, it goes ahead and continues with its own startup. Once this component begins consuming its own message queue, it will see its own "startup" messages. Since they contain this component's own unique ID, they can be safely discarded.

We send out *two* startup messages because of the possibility of a hung consumer. We've seen the problem in developer testing, but not in production. RabbitMQ thinks a process is still consuming RabbitMQ messages, but it's not really there. RabbitMQ delivers messages in round-robin fashion. So, in this case, it will "deliver" one message to the bad consumer, and the other message to the active consumer. The active consumer will respond "already running."

Control Queue

See Figure 5.

Figure 5. Work and Control Queues



Work and Control Queues

When the web server (or any BATS worker) is sending a message to a RabbitMQ exchange, that process configures the exchange. The exchange is *durable* meaning it should always exist except for the first time it's used. However, stuff happens, so everything using an exchange first declares the exchange. In RabbitMQ terms, this is a *passive* declaration, meaning if it already exists and the configuration parameters match the existing exchange, nothing happens. If the exchange does not yet exist, this declaration creates it.

When a worker starts up, it consumes from a *queue* and not directly from the exchange. The worker declares the queue and then binds it to the exchange. The binding includes a *routing key*. When a message is sent to the exchange with a matching routing key, RabbitMQ places a copy of the message into the queue. Copies of a given message could end up in one queue, multiple queues, or no queue.

When the message does not land in any queue, RabbitMQ discards the message. This means the worker needs to create and bind the queue before any message traffic can be sent to that queue.

Work queues, such as the queue to receive Pending Credits messages, are declared to be *durable*. This means the queue will continue to exist even if nothing is consuming that queue at the moment.

Control queues are configured as *expiring*. Once the worker stops, RabbitMQ automatically deletes the queue. The control exchange is a *fanout* exchange, which means any message sent to the exchange will be delivered to *all* queues bound to that exchange. Thus, to broadcast the "please register" message everywhere, the monitor only needs to send one "please register" message to the `bats_control` exchange.

Our BATS convention is to name queues according to the exchange and routing key. Thus the Pending Credits messages, with exchange `bats_realtime` and routing key `tv_events` are routed to a durable queue named `bats_realtime_tv_events`. This makes it easy to see the relationships between exchanges and queues on our RabbitMQ management pages.

RabbitMQ has a "vhost" system where different RabbitMQ users could be placed on different vhosts. However, when a worker is consuming multiple queues like we show here, all the queues need to be on the same vhost.

Second Iteration

We placed our Pending Credits worker into production. It immediately began processing 8-10 messages per second. We stopped the worker, let its RabbitMQ queue build up, and started the worker. The worker averaged about 50 messages per second while catching up.

So far, so good. A peak capacity of 5X the normal processing rate sounds perfect. But, wait, the whole point was to scale out for TV Channel traffic 12X current traffic. Let's back up and check the math.

Our traffic is higher during the day and lower at night. So let's do the math for 600,000 views over 20 hours. That's 30,000 views per hour, which translates to 30,000 Pending Credits messages per hour. Divide by 3,600 seconds per hour and we get 8.33 messages per second.

That's exactly our *observed* message rate our first day in production! It seems our TV Channel traffic had gone up dramatically since this project was conceived, but that fact was not fed back into the planned capacity.

Meanwhile, our system administrators assure me the BATS workers, RabbitMQ, and the local Redis instance can barely feel the load. We have plenty of room for scaling up.

Single Worker

The Pending Credits worker actually does a lot of other processing for statistical and reporting purposes. I decided to put everything into a single worker to avoid any concurrency issues. I did not need to worry about two things updating a record at the same time, because there is only one worker updating. I'd prefer to not change that design.

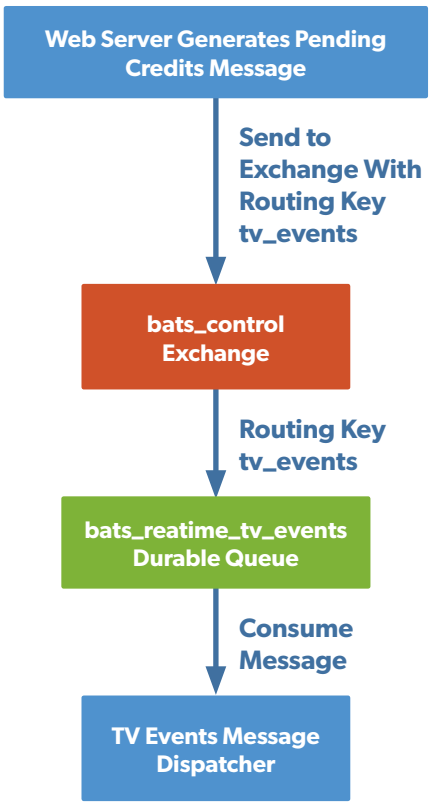
This means I can't simply throw more workers at the problem. They can potentially step on each other.

Multiple Workers

Instead we are *sharding* the work. If we want to run three workers, we take the message, examine the member ID, and assign it to the worker modulo three.

The web server doesn't know the difference. As we saw in Figure 1, the web server places the crediting information in Redis and a message containing the Correlation ID and Member ID in the `bats_realtime` exchange with routing key `tv_events`. Figure 6 shows the RabbitMQ portion of this.

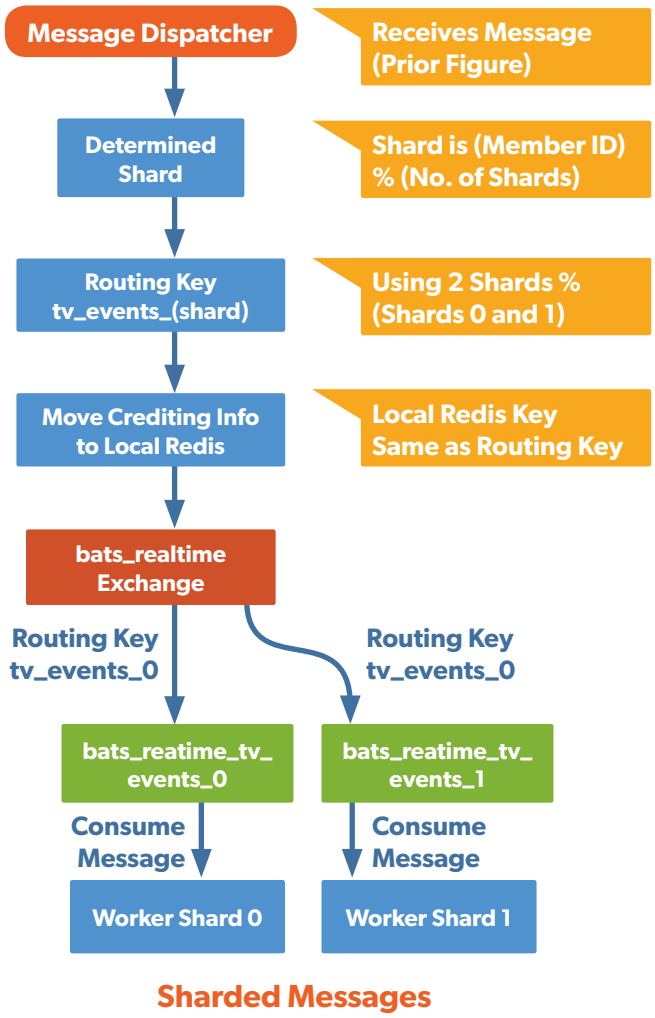
Figure 6. Message Receipt and Dispatch



We use a *Message Dispatcher*[6] to separate the message into shards. The message dispatcher receives the message in Figure 6 exactly like the TV Channel Worker did in Figure 5.

Figure 7 shows the message dispatcher routing the message to the correct TV Channel Worker:

Figure 7. Message Sharding



We've added a step. The message dispatcher routes messages to worker 0, worker 1, etc., for however many shards we have. This system allows us to run any number of workers. If we exceed the capacity of a single server, we could have the web server create a sharded routing key.

For example, if we have two BATS servers, the web server could use an even/odd routing key based on the member ID being even/odd. Each BATS server would still have a message dispatcher to route to shards 0, 2, 4, etc. or 1, 3, 5, etc.

Unlike sharding a database, here we have no difficulty rearranging the shards. It's only the transient message (both RabbitMQ and Local Redis) which is sharded. The ultimate destination is a non-sharded database.

6 *Enterprise Integration Patterns, page 508*

To change the number of shards, we need only stop the message dispatcher(s) and allow workers to drain their queues. The workers need not even be restarted, except to adjust for the new number of shards. Start the message dispatcher(s) with the new number of shards to use.

Beginning to Design for Scale

What is scale? With the TV Channel feature, our expected traffic increase represents a 10% increase in overall site traffic. That's an entire new web server, which is *not* part of our Strangler Pattern project.

When we begin to design for scale, we need to consider the choke points. We considered:

- Network congestion due to increased message (RabbitMQ server) traffic.
- Network congestion due to increased Redis Server traffic.
- UDP datagrams (memcache) which get "dropped on the floor" during overflow versus TCP packets (Redis) which trigger back-pressure mechanisms.
- Sheer data volume overwhelming our database storage.
- Additional load on our web servers.

We did our best to identify all monitoring needed to ensure we find out when something goes wrong. We expect to learn from ongoing observation. Martin Fowler[7] reports Google Fellow Jeff Dean[8] advises, "design for ~10X growth, but plan to rewrite before ~100X." Our Strangler Pattern system has been in production for some weeks, with three features using the system thus far. These features appear well-positioned for at least 10X growth.

Summary

This article describes our *Strangler Pattern* implementation. Even though we haven't shown any code yet, the diagrams do reflect our current production code.

We chose to *not* rewrite our primary web application. Instead, we're focusing on work which can be offloaded to BATS as a "fire and forget" operation. The Pending Credits system described here was part of the use case which justified our taking "time out" to develop our BATS infrastructure.

Our BATS system uses a combination of Redis storage and RabbitMQ messaging. We chose this combination to build in greater reliability and to allow for a "sharding" approach to running multiple workers in parallel.

In the upcoming articles of this series we'll actually write some code, and bring things full circle with Producer-Consumer programming examples.

Ed Barnard began his career with CRAY-1 serial number 20, working with the operating system in assembly language. He's found that at some point code is code. The language and details don't matter. It's the craftsmanship that matters, and that craftsmanship comes from learning and teaching. He does PHP and MySQL for InboxDollars.com.

7 *Martin Fowler:* http://martinfowler.com/bliki/SacrificialArchitecture.html

8 *Google Fellow Jeff Dean:* http://phpa.me/jeff-dean-wsdm09

What Are Interfaces, Abstracts, and Traits?

Chilion Snoek

In my work as a PHP developer I use a lot of the more "unknown" possibilities of PHP. Most of the time this works great, but the downside is none of my co-workers understand any of the code I write (not only because of the "unknown possibilities," but merely because I write rocket-science). In this article, I will try to take you on a journey through interfaces, abstract classes, and traits.

Interfaces

What Are Interfaces?

First, let me tell you a bit about what an interface is. An interface is a contract. A contract other code depends on, implements, and conforms to. It lets us rest well at night because we are sure any dependency implements the exact methods we expect. We can easily swap out one implementation for another. And with our brave new PHP 7 world which supports type hints we can even make sure the right parameter types are strictly used.

What Is, More Technically, an Interface?

Well, an interface[1] is a class —without all the functionality. There are two things you can do in an Interface:

- Extend other interfaces
- Define method signature(s)

Classes implementing the interface *must* implement the method(s) as defined.

Extend Other Interfaces

Extending an interface is much like extending a class. You can implement multiple interfaces, whereas you can only extend one class, as in Listing 1.

LISTING 1

```
01. <?php
02.
03. interface JohnnyFoo
04. {
05.     //there's no code in it!
06. }
07.
08. interface JonnyBar extends JohnnyFoo
09. {
10.     //this is not FUBAR.
11. }
```

1 *interface:* http://php.net/language.oop5.interfaces

Define Methods

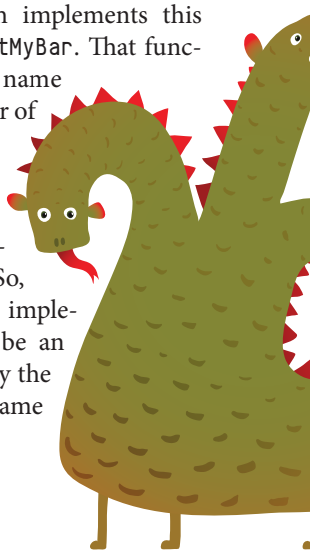
Let's take a look at defining methods. A method in an interface *must* be public. A private and/or protected function cannot be implemented. Also, a method in an interface can't have body; that part is for the class which implements the interface. This is where we get to the exciting part of the interfaces! See Listing 2 for a simple interface.

LISTING 2

```
01. <?php
02. namespace App\Interfaces;
03.
04. use App\Bar;
05.
06. interface JohnnyFoo
07. {
08.     public function getMyBar($id);
09.
10.     public function getMyOtherBar(Bar $bar);
11. }
```

What happens here? Well, the first function, `getMyBar($id)`, says every class which implements this interface must have a function called `getMyBar`. That function should always have a parameter; the name doesn't matter, as long as it has a parameter of the indicated type (if any).

The second method, `getMyOtherBar()`, expects an object instance of `App\Bar`. We declare at the top with the `use` declaration, this should be `App\Bar` instance. So, you should be sure that in any class that implements the interface there must always be an instance of an `App\Bar` object returned by the `getMyOtherBar()` method. Again, the name of the parameter doesn't matter at all.



Last, but not least a few extra points:

- A method in the implementing class can never have more or fewer parameters in the method declaration.
- An implementing class can have more methods than defined in the interface, but never fewer methods.
- If the interface uses type hints—including scalar hints in PHP 7 —for arguments and return hints (in PHP 7) then all implementations must conform exactly to the signature.

How Does Such a Class Look Then?

Well, let's see—check out Listing 3!

As you can see, the two methods from the interface are implemented, with the correct parameters and instances of parameters. I also have added a third function. Not with an actual function (pun intended), but just to show this is perfectly valid. (Well, except for the missing PHP DocBlocks; use them!)

Why and When Would I Use This?

Why is easy enough; to enforce code quality and abstraction. When? Every time you want to enforce code quality and abstraction! Abstraction helps keep your application flexible, since any object implementing an interface can be swapped in. This helps to make your code more testable.

On a more serious note, the when is almost always, but you should use `Interfaces` in an application where you have a lot of `Classes` which do the "same thing" but in a slightly different way. As an `Interface` is a contract, all of our `Classes` will look the same, feel the same, and do almost the same thing. Readability will increase enormously because everything looks the same. Everything works according to the pre-defined principle, (e.g., the contract). When you're injecting a dependency into another object, use an Interface as the type hint, instead of hinting to a concrete implementation of it.

LISTING 3

```
01. <?php
02.
03. namespace App\Classes;
04.
05. use App\Bar;
06. use App\Interfaces\JohnnyFoo;
07.
08. class Fubar implements JonnyFoo
09. {
10.     public function getMyBar($randomParameterName) {
11.         $baz = new Bar($randomParameterName);
12.         return $this->getMyOtherBar($baz);
13.     }
14.
15.     public function getMyOtherBar(Bar $againRandom) {
16.         // As long $againRandom an instance of Bar is,
17.         // lets check if they have something to drink.
18.         return $againRandom->drinks();
19.     }
20.
21.     public function setNonsense() {
22.         return 'demo function with no actual value';
23.     }
24. }
```

A Last Word About Interfaces

Interfaces aren't necessary before an application works, but they become more important every day when writing a larger application. The problem with working in teams on big applications is everyone writes code in their own way. Interfaces make sure you all code the same way, use the same methods, and implementation of architecture.

Abstract Classes

What Are Abstract Classes?

Abstract classes[2] are a commonly seen feature of Object-oriented programming. Abstract classes are a way to make sure your code quality and re-use increases and can drastically decrease the time spent writing duplicate or boilerplate code.

What Is, More Technically, an Abstract Class?

An abstract class is basically a blueprint. It is a class with methods, parameters, documentation, and—here comes the difference with an `Interfaces`—with basic functionality.

What Is the Big Difference With an Interface?

An `Interface` basically tells you what to do and gives you a contract with guidelines you have to follow quite strictly. An abstract class, on the other hand, tells you what to do and gives you some code to work with.

They also serve different purposes; when you use an abstract class, all child `Classes` are specific instances of that one base object (the abstract class). When you use an `Interface`, the `Class` implementing it is capable of doing something. The last difference might be the biggest—they

2 *Abstract classes:* http://php.net/language.oop5.abstract

LISTING 4

```php
<?php

namespace App\Abstracts;

abstract class AbstractBeverages
{
    /**
     * Determines the temperature of the beverage
     * @var $temperature
     */
    protected $temperature;

    /**
     * @return String
     */
    public function getTemperature() {
        return $this->temperature;
    }

    /**
     * @return mixed
     */
    abstract public function pour();
}
```

are implemented in a totally different way. An abstract class is extended from; but you can't extend more than one class. On the other hand, any class—including abstract ones—can implement multiple `Interfaces`.

What Does an Abstract Class Look Like, Then?

As an abstract class is a `Class`, let's take a look at Listing 4.

What happens here?

To get a beverage in our bar, we will have to create several `Classes`, all of which can extend our abstract class. Every beverage has a temperature, so we can easily make that code abstract. We don't have to repeat ourselves in every beverage `Class`.

First, we declare a `protected $temperature` and an accompanying method `getTemperature()`. Second, there is the `abstract public function pour()` which has no body at all, but because we declared it here, every class that extends it has to implement this function. Every developer that wants to use a beverage, has to pour it. Seems legit right? Otherwise, they would drink from the bottle. Yuck!

How Do We Use This, Then?

Let's pour a coffee (see Listing 5).

Now, let's add a Coke as well (Listing 6)!

As you can see, `Cola` and `Coffee` extend the `AbstractBeverage` class, which means all the child `Classes` will also inherit all the methods and properties of the abstract class. Within cola and coffee the pour function can be something completely different. Coffee returns an array to the developer, so he knows there is more to it. Cola, on the other hand, just tells us there is no cola available at the moment and we should pick beer.

LISTING 5

```php
<?php

namespace App\Classes;

use App\Abstracts\AbstractBeverages;

/**
 * Class Coffee
 * @package App\Classes
 */
class Coffee extends AbstractBeverages
{
    /**
     * Coffee constructor.
     *
     * @param $temperature in Celcius degree
     */
    public function __construct($temperature) {
        $this->temperature = $temperature;
    }

    /**
     * Lets pour some Coffee!
     *
     * @return array
     */
    public function pour() {
        return [
            "typeOfCup" => 1,
            "contains" => 300,
            "unit" => "ml",
            "temperature" => $this->temperature
        ];
    }
}

$coffee = new Coffee(76);
```

LISTING 6

```php
<?php

namespace App\Classes;

use App\Abstracts\AbstractBeverages;

/**
 * Class Cola
 * @package App\Classes
 */
class Cola extends AbstractBeverages
{
    /**
     * Cola constructor.
     *
     * @param $temperature in degrees Celsius
     */
    public function __construct($temperature) {
        $this->temperature = $temperature;
    }

    /**
     * Lets pour some Cola!
     *
     * @return string
     */
    public function pour() {
        return "We don't have Cola, we have beer.";
    }
}

$cola = new Cola(1);
```

When Do I Use This?

Now and then, you have certain bit of functionality used in multiple places. Boom, there you are! You reduce duplicate code and you make sure each extending `Class` is compatible with another, because you have the exact same logic.

A Real World Example

To elaborate more on our beverages, coffee, cola, beer, tea, milk, and water are all different types of beverages but share a lot in common; for instance, they're wet. So inheriting from an abstract class makes sense. To determine when to use it, you can ask yourself the question: "Is something a type of something?" Is a Honda Blackbird CBR 1100xx a type of Motorcycle? If so, structure its code around a `MotorcycleAbstract`. Amsterdam is a type of City (e.g., `CityAbstract`).

Last Few Words About Abstract Classes

The usage of abstract classes is important if you are building a large application and if you work in a team. Abstract classes are extremely easy to build and read, so they can accelerate your development process and help you and your teammates understand each other a little better every day. Also, reading other people's code will be problematic if you don't understand the abstract classes concept. Abstract classes are used very often in many open source packages.

Traits

To me, the biggest problems in PHP might be that you can't double, triple, or whatever-ple inherit. In other words, PHP only supports single inheritance, meaning one class can only inherit one other class.

This is, arguably, one of the weak points of PHP, but luckily the PHP crew decided to give us a new toy (in PHP 5.4 already) and they called it traits[3]!

What Is This?

A trait is nothing more than a grouped bunch of methods. On this point it is nothing more than a standard `Class`. There is more to it, but don't be afraid. It gives you the opportunity to mix traits into an existing `Class` to use methods of that particular trait. This gives us unlimited possibilities of using code over and over again, but not typing it over and over again without coupling it to any class!

- A trait is like an `Abstract Class`, not instantiable on its own.

How Do I Create a Trait?

It's not hard to create a trait, as they are just a collection of methods grouped together as mentioned before.

A simple file like Listing 7 is a trait.

LISTING 7

```
01. <?php
02.
03. namespace App\Somewhere;
04.
05. trait Motorcycles
06. {
07.     public function fuel() {
08.         return 'All motorcycles ride on gasoline';
09.     }
10.
11.     public function broooom() {
12.         return '*Make Noise!*';
13.     }
14. }
```

How Do I Use It?

You might have seen something like this before:

```
<?php

namespace App\Somewhere;

use App\Somethere\Something;

// ...
}
```

Well, that has nothing to do with traits. Except that the `use` word is used. And it's PHP, which counts as well, I guess. But I include it here to show that traits also re-use the `use` keyword.

```
<?php
namespace App\Somewhere

class Baz
{
   // "user" inside a class definition == trait
   use Motorcycles;
}
```

And that is how you use it. Or at least, `use` it. To dive in deeper, see Listing 8.

LISTING 8

```
01. <?php
02.
03. namespace App\Somewhere;
04.
05. class Baz
06. {
07.     use Motorcycles;
08.
09.     public function index() {
10.         return "something";
11.     }
12. }
13.
14. $baz = new Baz();
15. return $baz->broooom();
```

3 *Traits:* http://php.net/language.oop5.traits

As you can see, `Baz` does not have `broooom()` function, but through the trait, it does! In our case, `return $baz-> broooom();` returns a beautiful noise! (Because Honda, but that's logic.) So, in fact, a trait is just a copy-pasted chunk of code. But, we have only written it once; code duplication is gone! Because we `use` the trait in `Baz`, the moment it gets instantiated it will also have the method `broooom()` and the method `fuel()` for you.

This Sounds an Awful Lot Like Abstract Classes

They're not! A trait does not rely on inheriting anything. Imagine, the Honda CBR Blackbird 1100xx needs new tires. To inherit all the way (if you do it right) would be a hell of a job.

```
class tires extends s1100 {}
class s1100 extends Blackbird {}
class Blackbird extends CBR {}
class CBR extends Honda {}
class Honda extends Motorcycles {}
```

That's quite a lot inheritance in which we have the complete overload of inherited methods and properties which could be redundant. This is the `Abstract Classes` approach. Not totally wrong, but not great either.

With a trait, this is much easier to solve.

But What About an Interface? It Looks Like That, Too!

True, both are simple and when not actually implemented, useless. The difference between the two is *must* and *can*. An `Interface` is a contract telling you to, "let this `class` do this." A trait is more like, "Yo, this `class` can do this." A small example is in Listing 9.

LISTING 9

```
01. <?php
02. // Trait:
03. trait RabbitTrait
04. {
05.     public function color() {
06.         return 'none';
07.     }
08. }
09.
10. // Interface
11. interface RabbitInterface
12. {
13.     public function legs();
14.     public function ears();
15. }
16.
17. // Class
18. class Rabbit implements RabbitInterface
19. {
20.     use RabbitTrait;
21.
22.     public function legs() {
23.         //No code
24.     }
25.
26.     public function ears() {
27.         //No code
28.     }
29. }
```

The `Rabbit class` has the functions `legs`, `ears`, and `color`. But, the `Interface` forces me to put the methods in the `Class`. The trait just pastes them in it somewhere.

So, initializing above code in another file will work just fine:

```
<?php
$stamper = new Rabbit();

// Tell me, what color is Stamper?
echo $stamper->color();
```

Actual Benefits

traits will improve your code in the following ways:

- Code reuse, prevents code duplication
- Prevents complicated inheritance which doesn't make any sense
- Code better, faster

But, traits Have Some Problems!

But, be careful with traits!

Readability might get worse instead of better because you don't see all the methods in the source code anymore. You have to look deeper in to get to the actual code.

It's very easy to fall into the "this fits here" trap. Because it's so easy to share methods once, chances are you might create big, bloated, over the top classes that shouldn't exist and wouldn't in an environment without traits. Stay with the Single Responsibility Principle (It tells us a function/method/class/interface/trait should do one thing and one thing only. This makes sure that it's doing it well and not doing anything unexpected.)

A Real World Example of traits

It's no secret I'm a big fan of Laravel, so I picked a piece of code from its Eloquent package. Let's take a look at the `Notifiable` trait[4] which the `User` model uses. Besides a lot of other things, this `trait` gives the code in Listing 10.

LISTING 10

```
01. <?php
02.
03. namespace Illuminate\Notifications;
04.
05. trait HasDatabaseNotifications
06. {
07.     /**
08.      * Get the entity's notifications.
09.      */
10.     public function notifications() {
11.         return $this->morphMany(
12.                 DatabaseNotification::class, 'notifiable'
13.             )->orderBy('created_at', 'desc');
14.     }
15.
16.     /**
17.      * Get the entity's unread notifications.
18.      */
19.     public function unreadNotifications() {
20.         return $this->morphMany(
21.                 DatabaseNotification::class, 'notifiable'
22.             )->whereNull('read_at')
23.              ->orderBy('created_at', 'desc');
24.     }
25. }
```

Every Model which gets the `Notifiable` trait will have the function to see if there are notifications and if there are `unreadNotifications`. Which is awesome, because I don't only have users receiving notifications. I may have clients who get an email, but as soon as they read it, it sends back a push token that it has been read. Laravel uses traits extensively, `Validator` for Eloquent, for instance. Or my personal favorite, `SoftDeletes`.

Some Final Words on traits

traits are freaking awesome, if you use them correctly. Don't use them to cope with laziness! If you do that, you'll just have to refactor everything later. Most of the time rethinking your software architecture is better than using a trait. You can use traits on anything in your software, as long as it's a part of the code which is reused between `Classes` that don't inherit from the same `Abstract` or normal `Class`.

Final Notes

The big question here is, should I use `Interfaces`, `Abstract Classes`, and `traits`? The answer is definitely yes, but with one condition. Only in the right place, and at the right time. I hope your tool belt might have extended a bit with this article so you can continue writing code, but consistently better code!

Chilion is a young crazy developer that loves to write pragmatic code. He also is enthusiastic and definitely an extroverted person. He doesn't care if he is explaining something to 1 person or to a thousand, he just want to watch you learn and grow. Not only technically but also (and maybe even better) personally. He loves the Fast and Furious franchise, is crazy about his dog and wife (yes, in that order) and has a weird sense of humor. Laravel is his carport under which his CBR 1100 stands.

4 `Notifiable` *Trait:* http://phpa.me/laravel-notifiable-trait

Creating PHP Extensions With Zephir

Victor Bolshov

Creating C extensions for PHP has always been a challenge most PHP programmers were not able to accept. Some simply do not know C well enough, and others do not wish to learn the specifics of writing extensions involving a lot of C macros which are used throughout PHP's internal code. Zephir is a project which builds a bridge from C extensions-land to PHP user-land. You write your code in a high-level language which resembles PHP, get your Zephir code transpiled into a C extension which can be compiled using the standard PHP utility `phpize`, plus `configure`, and `make`. All that pointers hell is dealt with by Zephir build system, so you can sit back and relax having your PHP extensions created and maintained efficiently.

Too good to be true? Well, you're right; there are caveats. Yet, Zephir is without a doubt an interesting project which can be of a use to some developers. Maybe to you as well?

Zephir[1] is a relatively new programming language which allows for the creation of C extensions for PHP with the ease of writing in a high-level language. Zephir introduces a level of abstraction over PHP extensions API, which means your extensions written in Zephir are both PHP 7 and PHP 5 ready. C extensions are predictably faster than native PHP code on certain tasks, and in some cases your performance gain can be up to 200–300%—all by porting your PHP code to an extension with Zephir. The performance increase is much lower with PHP 7 than with PHP 5. When porting your code to an extension, you will also face new problems with debugging and deployment. You can follow development at the Zephir Github repo[2]

Introduction

Zephir—**Ze**nd Engine **Ph**p Int**r**mediate—is a high level language which eases creation and maintainability of extensions for PHP. Zephir extensions are exported to C code which can be compiled and optimized by major C compilers such as GCC/Clang/VC++. Functionality is exposed to the PHP language.

Zephir itself is written in PHP. What it does is take your `.zep` files with your source code and translates them into a C extension. Zephir is object-oriented, and classes you create in Zephir are exported to PHP. There's no way to create a procedural extension with Zephir. All the classes have to be namespaced. You can use **any PHP function** or class in your Zephir code, even if it is not a core PHP function. For example, you can create *PDO* instance or a *DOMDocument* if you need—the same way as you would in PHP (obviously, a relevant extension needs to be installed)! Once you're done with your new extension's code, Zephir will take care of building the C extension for you. To make an extension with Zephir, you don't need to know C or PHP's C internals, which in itself is a complicated subject. It opens the way to making high-performing, mission-critical code, for all PHP developers who do not have a strong C background.

What about big projects written in Zephir? Undoubtedly the most remarkable one is Phalcon[3], a full-featured framework for developing PHP applications, utilizing the MVC approach. Phalcon, currently in version 3.0.1, has a community around it and is a modern, robust framework which beats any competitor performance-wise.

Is Zephir PHP7-ready? The short answer is *yes*. Zephir supports both PHP 5 and PHP 7. There are some caveats if you have both PHP versions installed, though. Zephir is to a large extent associated with the Phalcon project, and I am glad to tell you Phalcon 3.0 runs on PHP 7!

Installation

Zephir is known to run on Linux, Mac OSX, and Windows. I will focus on Linux here, however, the steps are pretty much the same on Mac. Windows users, unfortunately, are going to

1 *Zephir:* https://zephir-lang.com

2 *Zephir Github repo:* https://github.com/phalcon/zephir

3 *Phalcon:* https://phalconphp.com

have a tough time following the instructions summarized at the Windows Installation Guide[4].

Packages you will need on Linux:

- **GCC** >= 4.x or Clang >;= 3.x
- **re2c** 0.13 or later
- GNU **Make** 3.81 or later
- **Autoconf** 2.31 or later
- **Automake** 1.14 or later
- **libpcre3**
- the **PHP development headers and tools**.

This is the list for Ubuntu. For other distros, package names might differ. Next, we install Zephir itself (I assume you have Git installed). Instructions for Ubuntu (except for exact package names), should be pretty much the same for the majority of Linux distros:

```
sudo apt-get install git gcc make re2c php5 php5-json \
  php5-dev libpcre3-dev autoconf automake
git clone https://github.com/phalcon/zephir
cd zephir
./install -c
```

The last command will ask for the **Sudo** password and install Zephir in `/usr/local/bin/zephir`. If you do not want it installed system-wide, just copy/symlink `bin/zephir` to a convenient location, probably within your `$PATH`. *The official documentation also mentions dependency on json-c package but this information is outdated.* Zephir installed smoothly on my machine with both PHP 5.6 and PHP 7.0.

4 *Windows Installation Guide:* https://github.com/phalcon/zephir/blob/master/WINDOWS.md

Note: *If, like me, you have several versions of PHP, and you switch between them now and then, it could be better to build your extensions with the option* `--parser-compiled=force`. *The Zephir parser, which is used internally by Zephir, is itself delivered as a PHP extension and that extension must be compiled with the same PHP version you are currently using.*

Writing the First Extension

Most likely, you will not want all of your PHP code converted to an extension. While this will give you some advantage in speed, it will not make your database queries or your network connections run faster. In most cases, the PHP code itself is not the most significant performance bottleneck. However, when it comes to CPU-intensive calculations, sometimes you can win a lot by using a C-extension. A good candidate for being ported to Zephir is a piece of code that either heavily uses the CPU or runs on nearly every request, so even a small gain might impact overall application performance.

Don't forget your deployment process will be different for the code running as an extension. You will need to adjust your build process, restart your web server or PHP-FPM, as opposed to deployment of regular PHP code, which can be done as easy as switching a symlink. Usually, you have to carefully choose which part of the system you are going to optimize by porting it to an extension.

For this article, I chose a somewhat artificial, yet demonstrative example: multiplying matrices.

Figure 1. Zephir command output

```
 _____                  __   _
/__  /  ___  ____  / /_  (_)____
  / /  / _ \/ __ \/ __ \/ / ___/
 / /__/  __/ /_/ / / / / / /
/____/\___/ .___/_/ /_/_/_/
         /_/

Zephir version 0.9.4a-dev-ba958bbc97

Usage:
        command [options]

Available commands:
        builddev            Generate/Compile/Install a Zephir extension in development mode
        stubs               Generates extension PHP stubs
        help                Displays this help
        clean               Cleans the generated object files in compilation
        generate            Generates C code from the Zephir code
        version             Shows the Zephir version
        install             Installs the extension (requires root password)
        fullclean           Cleans the generated object files in compilation
        compile             Compile a Zephir extension
        init [namespace]    Initializes a Zephir extension
        build               Generate/Compile/Install a Zephir extension
        api [--theme-path=/path][--output-directory=/path][--theme-options={json}|/path]Generates a HTML API

Options:
        -f([a-z0-9\-]+)     Enables compiler optimizations
        -fno-([a-z0-9\-]+)  Disables compiler optimizations
        -w([a-z0-9\-]+)     Turns a warning on
        -W([a-z0-9\-]+)     Turns a warning off
```

Project Skeleton

Now that you have Zephir installed, you should have `zephir` command available. Try executing it without arguments in your terminal, and you should see a fancy ASCII-art Zephir logo and the command usage reference. To create a project skeleton, type this in your shell:

```
zephir init phparch
```

The name of our new extension will be `phparch`. This will create the basic directory and file structure for our extension. `cd` to `phparch` folder, you will see two folders: `ext` and `phparch`. `ext` is where Zephir places the generated C code for the extension, `phparch` is the folder to place Zephir source files. Our Zephir project has a configuration in JSON format which you may find in `config.json`, in the root of your project. This JSON file syntax is pretty much straightforward, though most of its contents will only interest you once you master Zephir. For now, just check the `namespace`, `name`, `description`, `author`, and `version` entries.

Figure 2. Extension file structure

Zephir is only capable of creating object-oriented extensions. Moreover, classes you declare in your extension, have to be namespaced. Now, let's create our first class. Zephir class files, by convention, have a filename extension `.zep`, so let's create our `Matrix.zep` under the `phparch` folder:

```
namespace Phparch;
class Matrix {}
```

> **Note:** *namespace names in Zephir have to be capitalized, otherwise the compiler will raise an error.*

All the source code for this article is available on GitHub at crocodile2u/phparch-zephir[5]. Every major step we will do has a corresponding branch in that repository. Later on, I will refer to these specific steps as `⎇ *[branch name]*, like ⎇ *step1* that we're on at the moment.

> *You can clone the repo to follow along from:*
>
> ```
> git clone https://github.com/crocodile2u/
> phparch-zephir.git
> ```

As you see, this empty-class example has the same syntax as in PHP. It compiles, although it doesn't do anything. Let's go through the build procedure:

```
$ zephir build
Preparing for PHP compilation...
Preparing configuration file...
Compiling...
Installing...
[sudo] password for susieq:
Extension installed!
Add extension=phparch.so to your php.ini
Don't forget to restart your web server
```

You will be asked for a password to install `phparch.so` into PHP's *extension_dir*. We'll do all our tests at the command line. This is handy because you don't have to restart PHP-FPM or the web server, so you don't have to pay attention to the last line of the output. I assume you know how to find the `php.ini` file(s) loaded by your PHP-CLI. Add `extension=phparch.so` to your extension list, then we can check that everything works, that is, find our extension in the list of loaded modules:

```
$ php -m | grep phparch

phparch
```

If the output is empty, then something has gone wrong and the extension was not added to your PHP installation. Check the full output of the `php -m` command, make sure the extension is added to a relevant `php.ini` (there can be different `php.ini` files loaded for PHP running as CLI, PHP-FPM, and Apache). If that doesn't help, check error logs for the problem.

Now, let's see the extension information provided for us by PHP itself. If you run the following at the command line, you'll see output similar to Figure 3.

```
php --re phparch
```

Figure 3. Extension Info

```
Extension [ <persistent> extension #42 phparch version 0.0.1 ] {

  - Classes [1] {
    Class [ <internal:phparch> class Phparch\Matrix ] {

      - Constants [0] {
      }

      - Static properties [0] {
      }

      - Static methods [0] {
      }

      - Properties [0] {
      }

      - Methods [0] {
      }
    }
  }
}
```

Congratulations! You've just built your first PHP extension with Zephir! However, our *Matrix* class is empty and does nothing (a bug-free class, whoah!). Now, I'll continue with explaining how to really write code in Zephir, and the differences between it and PHP.

5 *crocodile2u/phparch-zephir:* https://github.com/crocodile2u/phparch-zephir

It's Compiled!

Yes, the code you write in Zephir needs to be compiled to run. In fact, Zephir first makes a "precompilation," checks your code against Zephir syntax and type rules, then, if the checks pass, transpiles the code to C and, after that, compiles it. Compilation, obviously, adds an extra step and takes some extra time (well, Zend Engine also compiles your PHP code into bytecode, but that process is somewhat hidden from the user's eye). Let's put in a few lines of code to our extension:

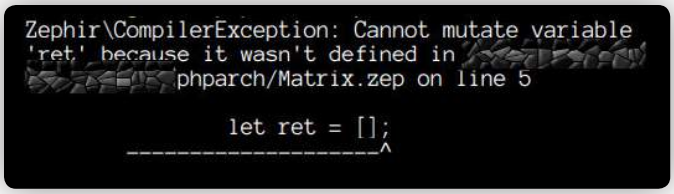
```
namespace Phparch;
class Matrix {
   public static function multiply(array a,
                                   array b) -> array
    {
        let ret = [];
        return ret;
    }
}
```

Build…

```
zephir build
```

You should see the error message in Figure 4.

Figure 4. CompilerException



You might have noticed the keyword `let` preceding the assignment. That is how variable mutations occur in Zephir, but the engine complained about `ret` variable not being defined. To effectively transpile from Zephir to C all variables need to be declared first:

```
var ret;
let ret = [];
return ret;
```

Compiler warnings are good because they point you to the exactly where the problem is. Now the updated class compiles, though it obviously doesn't do what we want. Moving further with Zephir programming, you will quickly find it is a bit harder than with PHP to make sure your code does the right thing. When I write in PHP, I often use Xdebug to get inside the running app and see where the execution goes, step by step. With Zephir, it's a different story.

Getting Things Done Right

If you try to debug Zephir code with, say, Xdebug, you will not succeed, because you'll want to debug an internal method implemented in C rather than in PHP. Those are black boxes for Xdebug. Of course, you may put your own debug symbols here and there: `var_dump` or `print_r` in particular. While this can be helpful, especially on early stages of your Zephir project, it gets annoying, and it results in the possibility of releasing your code with all the debug output accidentally left there. And this is where your good old friend appears. Ladies and gentlemen: PHPUnit!

```
use Phparch\Matrix;

class MatrixTest extends \PHPUnit_Framework_TestCase
{
   /**
    * @expectedException \TypeError
    */
   function testMultiplyThrowsWhenArgumentsAreNotArrays() {
      Matrix::multiply(1, 2);
   }
}
```

See *step2*. We're not focusing on PHPUnit in this article, so I will only rarely mention new test cases further. However, you're encouraged to check them out. My practice with Zephir shows it is extremely useful to limit the functionality of the extension and to unit test everything. Because classes in the extension are exposed to PHP like any other built-in classes, you can always use your mocking library in your tests to create mocks and stubs.

Zephir Basics

The algorithm for matrix multiplication requires that matrix A has exactly as many columns as matrix B has rows. We'll probably need to know the size of each matrix and we'll have to be sure all the elements in them are numbers. For that, I decided to add another method to our Matrix class: `getDimensions()`:

This method does a bit too much to be considered an example of good design, but it's still small enough to be readable. You can see not only does it return the number of rows and columns in the input array but also ensures it conforms to a specific format: all elements are numbers, all rows are of equal non-zero length. And, yes, we have it unit-tested! Notice the usage of PHP's built-in `InvalidArgumentException` class—just as you would in your PHP code. I am also making use of class constants for specific error codes like `ERR_ROW_EMPTY`. Those are declared on top of the `Matrix` class and look just like PHP class constants. Zephir lacks a `self` keyword, thus to reference a constant, you will have to use it with a class name, even within the same class. The code here also uses a *for* loop, but I will focus on loops a bit later.

```
const ERR_ARG_EMPTY = 1;
```

When you write your PHP code which uses the extension (like the PHPUnit tests in our case), you will probably want code autocomplete features in your IDE. Zephir offers you a nice command line utility *zephir stubs*. Run it in the Zephir project root, and it will add an `ide` folder, which contains PHP class stubs generated from Zephir files. In your IDE, add this folder to the `include_path` for your project, then all the classes and methods from your Zephir extension will be available to autocomplete. See *step3* and Listing 2 and 3.

LISTING 1

```
// @param int[][] input
// @return [int rowCount, int columnCount]
public static function getDimensions(array input) -> array
{
    var rowCount;
    let rowCount = count(input);
    if 0 == rowCount {
        throw new \InvalidArgumentException(
            "Argument is empty", Matrix::ERR_ARG_EMPTY
        );
    }
    var row, columnCount, rowColumnCount, i;
    let columnCount = null;
    for i, row in input {
        if !is_array(row) {
            throw new \InvalidArgumentException(
                "Row " . i . " is not an array",
                Matrix::ERR_ROW_NOT_ARRAY
            );
        }
        let rowColumnCount = count(row);
        if null === columnCount {
            // First row. All the other rows will have
            // to be of this length.
            let columnCount = rowColumnCount;
        }
        if 0 == rowColumnCount {
            throw new \InvalidArgumentException(
              "Row " . i . " is empty",
              Matrix::ERR_ROW_EMPTY
            );
        }
        if rowColumnCount != columnCount {
            throw new \InvalidArgumentException(
                "Row " . i . " length differs from "
                . rowColumnCount,
                Matrix::ERR_ROW_LENGTH_DIFFERS
            );
        }
        var col, j;
        for j, col in row {
            if !is_numeric(col) {
                throw new \InvalidArgumentException(
                    "Element at row " . i . ", column "
                    . j . " is not a number",
                    Matrix::ERR_NOT_A_NUMBER
                );
            }
        }
    }
    return [rowCount, columnCount];
}
```

LISTING 2

```
namespace Phparch;

class Matrix
{
    const ERR_ARG_EMPTY = 1;
    const ERR_ROW_NOT_ARRAY = 2;
    const ERR_ROW_EMPTY = 3;
    const ERR_ROW_LENGTH_DIFFERS = 4;
    const ERR_NOT_A_NUMBER = 5;

    public static function multiply(array a,
                                    array b) -> array {
        var ret;
        let ret = [];
        return ret;
    }

    // @param int[][] input
    // @return [int rowCount, int columnCount]
    public static function getDimensions(array input) -> array
    {
        var rowCount;
        let rowCount = count(input);
        if 0 == rowCount {
            throw new \InvalidArgumentException(
                "Argument is empty", Matrix::ERR_ARG_EMPTY
            );
        }

        var row, columnCount, rowColumnCount, i;
        let columnCount = null;
        for i, row in input {
            if !is_array(row) {
                throw new \InvalidArgumentException(
                    "Row " . i . " is not an array",
                    Matrix::ERR_ROW_NOT_ARRAY
                );
            }
            let rowColumnCount = count(row);
            if null === columnCount {
                // First row. All the other rows will
                // have to be of this length.
                let columnCount = rowColumnCount;
            }
            if 0 == rowColumnCount {
                throw new \InvalidArgumentException(
                    "Row " . i . " is empty",
                    Matrix::ERR_ROW_EMPTY
                );
            }
            if rowColumnCount != columnCount {
                throw new \InvalidArgumentException(
                    "Row " . i . " length differs from "
                    . rowColumnCount,
                    Matrix::ERR_ROW_LENGTH_DIFFERS
                );
            }
            var col, j;
            for j, col in row {
                if !is_numeric(col) {
                    throw new \InvalidArgumentException(
                        "Element at row " . i . ", column "
                        . j . " is not a number",
                        Matrix::ERR_NOT_A_NUMBER
                    );
                }
            }
        }
        return [rowCount, columnCount];
    }
}
```

LISTING 3

```
<?php

use Phparch\Matrix;

class MatrixTest extends \PHPUnit_Framework_TestCase
{
    /**
     * @expectedException \TypeError
     */
    function testMultiplyThrowsWhenArgumentsAreNotArrays() {
        Matrix::multiply(1, [2]);
    }

    function testGetDimensionsThrowsWhenArgumentIsEmpty() {
        $this->setExpectedException(
            \InvalidArgumentException::class,
            "", Matrix::ERR_ARG_EMPTY
        );
        Matrix::getDimensions([]);
    }

    function testGetDimensionsThrowsWhenARowIsNotArray() {
        $this->setExpectedException(
            \InvalidArgumentException::class,
            "", Matrix::ERR_ROW_NOT_ARRAY
        );
        Matrix::getDimensions([1]);
    }

    function testGetDimensionsThrowsWhenARowIsEmpty() {
        $this->setExpectedException(
            \InvalidArgumentException::class,
            "", Matrix::ERR_ROW_EMPTY
        );
        Matrix::getDimensions([[]]);
    }

    function testGetDimensionsThrowsWhenRowsHaveDifferentLength() {
        $this->setExpectedException(
            \InvalidArgumentException::class,
            "", Matrix::ERR_ROW_LENGTH_DIFFERS
        );
        Matrix::getDimensions([[1], [1, 2]]);
    }

    function testGetDimensionsThrowsWhenElementIsNotNumeric() {
        $this->setExpectedException(
            \InvalidArgumentException::class,
            "", Matrix::ERR_NOT_A_NUMBER
        );
        Matrix::getDimensions([[1], ["non a number"]]);
    }

    /**
     * @param int[][] $input
     * @param int[] $expectation
     * @dataProvider providerTestGetDimensionsOnValidInput
     */
    function testGetDimensionsOnValidInput($input, $expectation) {
        $result = Matrix::getDimensions($input);
        $this->assertEquals($expectation, $result);
    }

    function providerTestGetDimensionsOnValidInput() {
        return [
            "1x1" => [
                [
                    [60],
                ],
                [1, 1]
            ],
            "2x3" => [
                [
                    [66, 66, 66],
                    [67, 67, 67],
                ],
                [2, 3]
            ],
        ];
    }
}
```

Completing the Matrix Extension

Now it's time to make our extension do the job it was made for. Let's implement the *multiply()* method (and of course, you never forget about unit tests, do you?):

LISTING 4

```
public static function multiply(array a,
                                array b) -> array {
    var ret, aDimensions, bDimensions;
    let ret = [];
    let aDimensions = self::getDimensions(a);
    let bDimensions = self::getDimensions(b);
    if aDimensions[1] != bDimensions[0] {
        throw new \InvalidArgumentException(
            "The number of columns on matrix A is not"
          . "the same as the number of rows of matrix B",
            Matrix::ERR_NOT_MULTIPLIABLE
        );
    }
    var i, aRow;
    for i, aRow in a {
        var j;
        let j = 0;
        while j < bDimensions[1] {
            var k, ij, aCell;
            let ij = 0;
            for k, aCell in aRow {
                let ij += aCell * b[k][j];
            }
            let ret[i][j] = ij;
            let j += 1;
        }
    }
    return ret;
}
```

The code is pretty much straightforward. As promised, I will pay some attention to cycles. The `for i, aRow in a` construct is merely an equivalent of PHP's `foreach ($a as $i => $aRow)`. Strangely enough, Zephir does not have a traditional `for` loop like PHP does `for (expr1; expr2; expr3)`.

In the example above, the cycle inside the first `for` loop is `while`. I chose it for two reasons:

1. shows different types of cycles, and
2. with `for` I would be stuck to using a weird construct, such as

```
for j in range(0, bDimensions[1] - 1)
```

Our code compiles and our test can run. See *step4* and Listing 5 and Listing 6 in the downloadable code archive or on the GitHub repo[6].

But, we wanted Zephir not just to study yet another language. Yeah, it's fun to be able to easily write extensions for PHP, thus feeling closer to the heart of the language. But what we really wanted is performance! Well, let's see what our gain is.

6 *GitHub repo:* https://github.com/crocodile2u/phparch-zephir/tree/step4

Performance Comparison

I created a version of matrix class in pure PHP to compare performance. If you cloned the Git repo with the example, you will find it in `performance-test` folder next to the `phparch` which contains our Zephir project. See Listing 7.

LISTING 7

```
<?php
class Matrix {
    const ERR_ARG_EMPTY = 1;
    const ERR_ROW_NOT_ARRAY = 2;
    const ERR_ROW_EMPTY = 3;
    const ERR_ROW_LENGTH_DIFFERS = 4;
    const ERR_NOT_A_NUMBER = 5;
    const ERR_NOT_MULTIPLIABLE = 6;

    static function multiply(array $a, array $b) {
        $ret = [];
        $aDimensions = self::getDimensions($a);
        $bDimensions = self::getDimensions($b);
        if ($aDimensions[1] != $bDimensions[0]) {
            throw new \InvalidArgumentException(
                "The number of columns on matrix A is not"
            . " the same as the number of rows of matrix B",
                self::ERR_NOT_MULTIPLIABLE
            );
        }
        foreach ($a as $i => $aRow) {
            for ($j = 0; $j < $bDimensions[1]; $j++) {
                $ij = 0;
                foreach ($aRow as $k => $aCell) {
                    $ij += $aCell * $b[$k][$j];
                }
                $ret[$i][$j] = $ij;
            }
        }
        return $ret;
    }

    /**
     * @return array [int rowCount, int columnCount]
     */
    public static function getDimensions(array $input)
    {
        $rowCount = count($input);
        if (0 == $rowCount) {
            throw new \InvalidArgumentException(
                "Argument is empty", self::ERR_ARG_EMPTY
            );
        }
        $columnCount = null;
        foreach ($input as $i => $row) {
            if (!is_array($row)) {
                throw new \InvalidArgumentException(
                    "Row " . $i . " is not an array",
                    self::ERR_ROW_NOT_ARRAY
                );
            }
            $rowColumnCount = count($row);
            if (null === $columnCount) {
                $columnCount = $rowColumnCount;
            }
            if (0 == $rowColumnCount) {
                throw new \InvalidArgumentException(
                    "Row " . $i . " is empty",
                    self::ERR_ROW_EMPTY
                );
            }
            if ($rowColumnCount != $columnCount) {
                throw new \InvalidArgumentException(
                    "Row " . $i . " length differs from "
                    . $rowColumnCount,
                    self::ERR_ROW_LENGTH_DIFFERS
                );
            }
            foreach ($row as $j => $col) {
                if (!is_numeric($col)) {
                    throw new \InvalidArgumentException(
                        "Element at row " . $i . ", column "
                        . $j . " is not a number",
                        self::ERR_NOT_A_NUMBER
                    );
                }
            }
        }
        return [$rowCount, $columnCount];
    }

    static function dump(array $a) {
        $m = null;
        foreach ($a as $row) {
            $rowMax = max($row);
            if ((null === $m) || $rowMax > $m) {
                $m = $rowMax;
            }
        }
        $mlen = strlen($m);
        foreach ($a as $row) {
            foreach ($row as $col) {
                printf("% {$mlen}d  ", $col);
            }
            echo "\n";
        }
    }
}
```

LISTING 8

```php
<?php

$cycleCount = 1000;

$aRowCount = 10;
$aColCount = 12;
$bRowCount = 12;
$bColCount = 10;

function random_matrix($rowCount, $colCount) {
    $ret = [];
    for ($i = 0; $i < $rowCount; $i++) {
        for ($j = 0; $j < $colCount; $j++) {
            $ret[$i][$j] = rand(0, 100);
        }
    }
    return $ret;
}

$a = random_matrix($aRowCount, $aColCount);
$b = random_matrix($bRowCount, $bColCount);

function accept_input($name) {
    echo "Enter matrix $name:\n";
    $ret = [];
    while ($line = fgets(STDIN)) {
        if (!trim($line)) {
            return $ret;
        }
        $ret[] = array_map(
           "intval", explode(" ", $line)
        );
    }
}

// uncomment if you want to enter input matrices
// manually instead of generating random ones
//$a = accept_input("A");
//$b = accept_input("B");

$start = microtime(true);
for ($i = 0; $i < $cycleCount; $i++) {
   $resultZephir = \Phparch\Matrix::multiply($a, $b);
}
$end = microtime(true);
$elapsedZephir = ceil(($end - $start) * 1000);

echo "\n";
echo "ZEPHIR: $elapsedZephir ms elapsed\n";

require_once __DIR__ . "/Matrix.php";

$start = microtime(true);
for ($i = 0; $i < $cycleCount; $i++) {
   $resultPhp = \Matrix::multiply($a, $b);
}
$end = microtime(true);
$elapsedPhp = ceil(($end - $start) * 1000);
echo "PHP: $elapsedPhp ms elapsed\n";

echo "PHP implementation is "
   . round($elapsedPhp / $elapsedZephir, 2)
   . " times slower than Zephir\n";

if (!getopt("i")) {
   exit;
}

echo "\n";
echo "Inputs and results:\n";

echo "Input A:\n";
\Matrix::dump($a);

echo "Input B:\n";
\Matrix::dump($b);

echo "Zephir:\n";
\Matrix::dump($resultZephir);

echo "PHP:\n";
\Matrix::dump($resultPhp);
```

PHP implementation is a direct port of Zephir, with only minor changes. And, I have also written a small script which you can find in the same folder, called `compare.php` (see Listing 8).

It generates two random matrices, then multiplies them 1000 times using both implementations. On my box, with PHP 7.0.4, I have the following results:

```
ZEPHIR: 49 ms elapsed
PHP: 76 ms elapsed
PHP implementation is 1.55 times slower than Zephir
```

Not bad! And if you have PHP 5, then the performance gain should be noticeably higher (results for PHP 5.6 on the same computer):

```
ZEPHIR: 80 ms elapsed
PHP: 224 ms elapsed
PHP implementation is 2.8 times slower than Zephir
```

Almost three times faster with Zephir! Impressive, isn't it?

Conclusion

Zephir is worth taking a look at if you're really into PHP and have some code which needs heavy optimization CPU-wise. It's easy to start programming in Zephir if you're already familiar with PHP. You can get a significant performance boost converting bottlenecks to a C extension, yet having it written in a high-level language so it is easier to maintain. Nonetheless, it will still be another technology in your stack, which adds complexity to development and deployment process. Zephir is somewhat hard to debug, so you'd want all the Zephir code to be 100 percent covered with unit tests. Happy coding, folks!

Victor Bolshov (born in the USSR) studied at the Obninsk State University for Nuclear Power Engineering, where he got a master's degree in physics. Since 2001 Victor has been working as a web developer. His major field of interest is high-loaded, scalable applications, their architecture and optimization. He used to be a speaker at PHP-related conferences: PHPConf in 2007 and DevConf in 2016—in Moscow. You can find Victor on Twitter and GitHub under the nickname of @crocodile2u.

Celebrating 25 Years of Linux!

ORDER NOW!

Get 7 years of *Ubuntu User*

FREE

with issue #30

***Ubuntu User* is the only magazine for the Ubuntu Linux Community!**

BEST VALUE: Become a subscriber and save 35% off the cover price!
The archive DVD will be included with the Fall Issue, so you must act now!

Order Now! Shop.linuxnewmedia.com

Dev Divas: History's Heroines of Computing, Part Two

Vesna Vuynovich Kovach

In the previous installment of this series, I talked about some of the women who made foundational contributions to the world of computing. This month, I'll share stories about women in some of history's greatest computer collaborations, and the impact they had on our world.

Log page with moth
By Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. Public domain, via Wikimedia Commons

Grace Hopper

Last month, we met Rear Admiral Grace Murray Hopper, who coordinated the multidisciplinary creation of COBOL[1]. In 1954, as chairman of the Association of Computing Machinery's Committee on Nomenclature, she produced the *First Glossary of Programming Terminology*[2], standardizing the use of words which "have acquired an added meaning" particular to computer programming. Terms like *command*, *random access*, *function*, and even *debug* are found here. This last is particularly fitting; back in 1947, her programming team famously *taped into their log book a moth* that had brought their Harvard Mark II to a noisy halt, and jokingly dubbed it the "first actual case of bug being found." The log page is now in the *Smithsonian's collection*[3].

The PARC Legacy

We also learned about Adele Goldberg, who ran the Xerox PARC lab which generated and refined many ideas, such as the windowing metaphor, that today are woven into just about every personal computing interaction. She wasn't the only woman doing important work at PARC in the 1970s and 1980s. *Color scientists Maureen Stone and Bernice Rogowitz*[4] brought order and fidelity to color image editing and screen-to-print technology. *Lynn Conway*[5] did seminal work on microchips with Carver Mead; if you have a tiny computer in your pocket or on strapped onto your wrist, you have their "Mead & Conway Revolution in VLSI" Very Large Scale Integration) to thank. Lynn has contributed to workplace equality as well as to tech, as chronicled in *Life, Engineered: How Lynn Conway Reinvented Her World and Ours*[6] and other articles. In the late 1990s, Lynn revealed the secret she'd kept for nearly twenty years: she had been fired from a successful career at IBM in 1968 when she announced her intention to transition to a female gender role. A year later, she reentered the workforce under a new "stealth" identity. Since coming out as a transgender woman, she's been an LGBT activist, and was instrumental in bringing about the *2014 inclusion of gay and trans rights in the IEEE code of ethics*[7].

The ENIAC Women

I also promised I'd tell more about Frances "Betty" Snyder Holberton, the scion of a Philadelphia Main Line family of intellectuals which included astronomers and a founder of the National Bureau of Standards. As part of a team that invented programming as we know it, Betty worked on the world's first-ever electronic computer, ENIAC.

When I started trying to find out what women had

1 *the multidisciplinary creation of COBOL:* http://phpa.me/mit-grace-hopper

2 *First Glossary of Programming Terminology:* http://phpa.me/first-glossary-programming-terms-pdf

3 *Smithsonian's collection:* http://phpa.me/log-book-bug

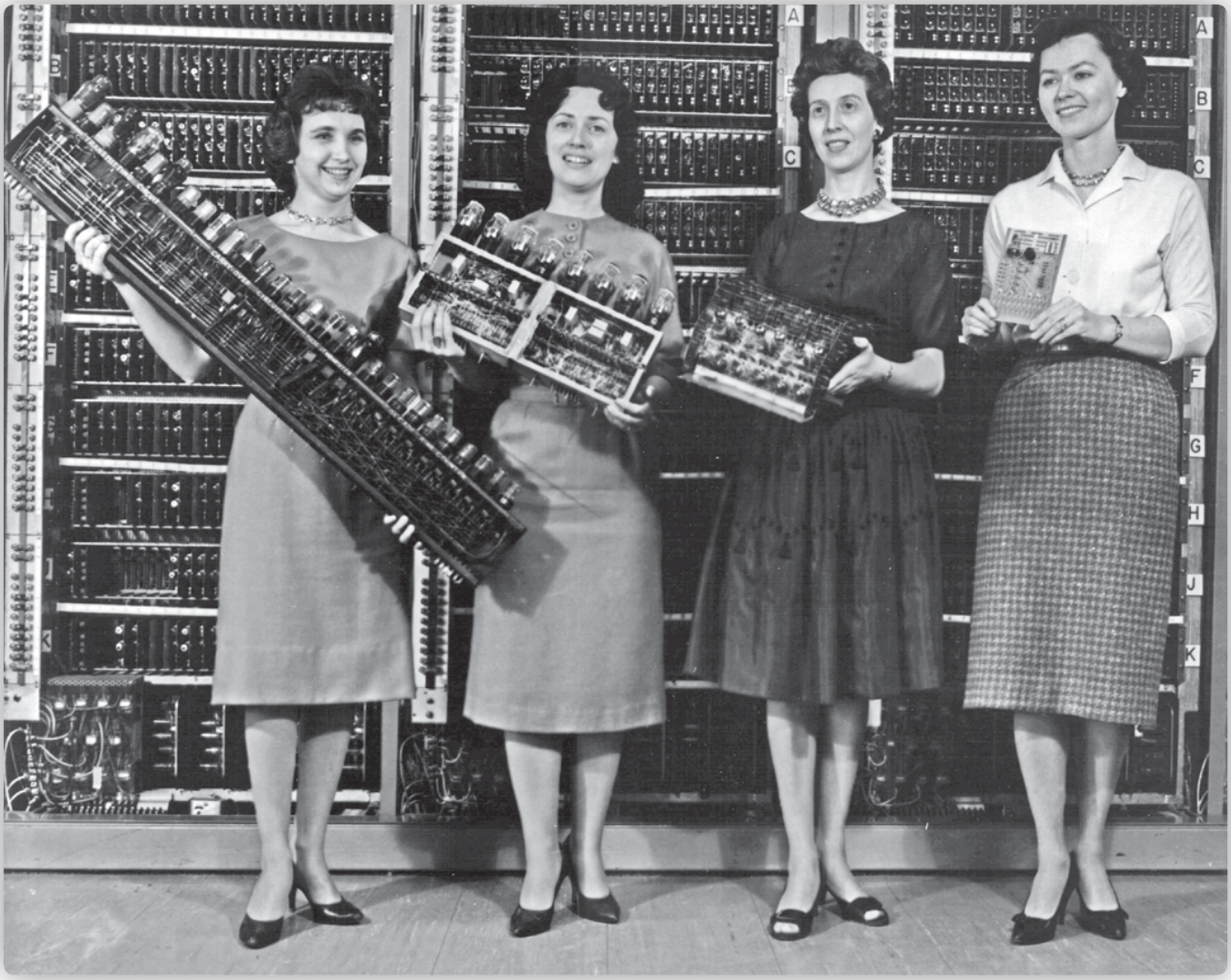
4 *Color scientists Maureen Stone and Bernice Rogowitz:* http://phpa.me/fastcodesign-parc

5 *Lynn Conway:* http://phpa.me/latimes-gender-labyrinth

6 *Life, Engineered: How Lynn Conway Reinvented Her World and Ours:* http://dme.engin.umich.edu/lynnconway/

7 *2014 inclusion of gay and trans rights in the IEEE code of ethics:* http://www.washingtonblade.com/?p=84424

The ENIAC Women



accomplished in computing history, the story of the ENIAC programmers was the first one I went looking for; though at first, I didn't even know it by that name. All I had to go on was a slip of memory which had followed me for years.

I remembered seeing a fragment of a documentary on cable TV showing some black-and-white photographs of an early computer during the WWII era. Young women appeared in these photos, posed before an electronic behemoth towering over them, enveloping them in seemingly endless dark walls of circuitry. The women stood straight or crouched down, grasping plugs on the ends of long wires, poising them against jacks set among the computer's byzantine array of switches and lights. Some rested their fingertips on dials as if about to twist them. One looked down at the papers in her hands, turned towards her coworker as if about to read out some plugging-in, dial-twisting instructions.

With their hair swept into elaborate 1940s rolls and pin curls, their elegant (although low) heels and long skirts, and their apparent preoccupation with moving plugs around, the women looked all the world like old-fashioned telephone switchboard operators—like comedian Lily Tomlin's signature character, the cheeky "Ernestine." Just with a much bigger switchboard.

This documentary explained these women had been misidentified for years. Some photo captions described them as data-entry technicians. Sometimes they were even thought to have been models, posed in the picture just to make the computer look good. Like pinup girls on automobile shop calendars.

In fact, *these women were the computer's programmers.*

I couldn't find that documentary, but I found plenty of information about the women in those photos. Much of their prominence today—through awards and other professional recognition, Wikipedia pages, YouTube videos, transcripts of oral histories, print and online articles, books—rippled from decades of effort by one Kathy Kleiman, a computer scientist who has made it her life's work to bring recognition to the six pioneers often called the "ENIAC women," or sometimes the "ENIAC girls." Kleiman's award-winning 2014 documentary, *The Computers: The Remarkable Story of the ENIAC Programmers,*[8] became available for viewing *on Vimeo*[9] in April 2016, after years of exclusive

8 *The Computers: The Remarkable Story of the ENIAC Programmers,*: http://eniacprogrammers.org

9 *on Vimeo:* https://vimeo.com/ondemand/eniac6

film festival screenings. Documentarian LeAnn Erickson's 2010 film, *Top Secret Rosies: The Female Computers of WWII*[10], also precipitated a cascade of discovery of information that had very nearly been lost forever.

Here's what I learned.

In the heat of WWII, the U.S. Army Ordnance Corps needed a faster way to compute ballistics trajectories. Gunners consulted tables filled with these calculations so they could aim long-range weapons firing at targets over a mile away. The Army had in its employ 176 computers—that is to say, humans who performed calculations, the meaning of that word for centuries—working overtime to compile these desperately needed tables.

Adele Goldstine

Mathematician Adele Goldstine had scoured universities and schools across the U.S. to build this crack team. Nearly everyone she hired was a woman—the men had gone to fight the war. Most of these women were degreed mathematicians, qualified to work out the differential calculus equations needed to build the firing tables. After she couldn't find any more mathematicians, she started hiring talented women and taught them advanced calculus herself. But, even with the aid of adding machines and a complicated single-purpose mechanical device called a "differential analyzer," the human computers couldn't keep up with the demand.

In 1943, Adele's husband, Lieutenant Herman Goldstine, also a mathematician, talked the Army into funding the Electronic Numerical Integrator and Calculator, or ENIAC. The first electronic, reprogrammable, stored-program computer ever built, ENIAC was the brainchild of two men at the University of Pennsylvania, physicist John Mauchly and an engineering grad student, Presper Eckert. In 20 seconds, they hoped, ENIAC would do a week's worth of human computing.

By the summer of 1945, two years and half a million dollars later, as far as anyone knew, the hard work was over. The computer was built. Everyone involved believed all that remained was an easy, uncreative, unglamorous chore, one which would be dispatched fairly quickly and with finality: figuring out exactly how to tell the computer what to do.

So they got women to do it.

Adele and Herman recruited a handful of women from the ranks of the human computers; they seemed a logical choice to feed the equations to ENIAC. Years later, one of those women, Betty "Jean" Jennings Bartik, would write in her autobiography, "If the ENIAC's administrators had known how crucial programming would be to the functioning of the electronic computer and how complex it would prove to be, they might have been more hesitant to give such an important role to women."

In fact, no one, not the Goldstines and certainly not the ENIAC's inventors, Eckert and Mauchly, anticipated the difficulty of getting the ENIAC to perform.

The newly selected "coders" went to Maryland for a few weeks to learn about punch cards and "plugboards"—those switchboard-looking things in the photographs. At Aberdeen Proving Grounds, Minerva Masoncup was in charge of the IBM punch card installation, the same technology which would be used for ENIAC's data input and output. She gave the women experience with plugboards jammed with as many as 600 electrical-connector holes.

That was the only instruction the ENIAC women got.

Back at the University of Pennsylvania's Moore School of Engineering, the women were handed a stack of diagrams, massive papers three by four feet in size. These were the circuit diagrams for ENIAC. They got one more thing: an assignment. They needed to start calculating missile trajectories.

The women still hadn't seen the machine or met its inventors. Nor were they provided an office or any other dedicated workplace. They dragged the diagrams into some empty classrooms where they could lay them out to decipher the unfamiliar notation and make sense of what they saw.

Though untrained in electrical engineering, they gradually made out the workings of the thing. It had an initiating unit, a master programmer, several accumulators, a constant transmitter, a square rooter, and a multiplier. The women recognized the unprecedented potential of this tool. The stepper switches allowed them to construct loops. They could use the positive/negative switch to navigate if/then conditions. They could even run several operations in parallel. They could harness the power of their algorithms.

And, during that sweltering Philadelphia July, amid the roar of building construction from the floor above, this truly agile team of developers invented the art of computer programming.

By February, they could debug the computer down to an individual vacuum tube—of which there were nearly 20,0000. But more than that, they'd invented programming routines, along with systems for creating, organizing and documenting them.

The war was over before the army got trajectories run, but as these women understood this wasn't just a trajectory calculator. It was a world-changing tool. A general-purpose computer. ENIAC remained in service for another ten years.

One of the programmers was the aforementioned Jean Bartik, a 21-year-old math major from rural Missouri whose programming career lasted into the 1980s. She trained Adele Goldstine to program, and the two of them spent the summer of 1946 working on the first extensive ENIAC program, one which concerned the physics of shock waves. By this time, Adele had written the excellent *technical manual for the ENIAC*[11] (a handsome *2012 Army reprint*[12] of the 472-page book is avail-

10 *Top Secret Rosies: The Female Computers of WWII:* http://sites.temple.edu/topsecretrosies/

11 *technical manual for the ENIAC:* http://phpa.me/46eniac-report

12 *2012 Army reprint:* https://www.amazon.com/dp/1937684660

able for sale online), and her intimate knowledge of the system made her a quick study. Their work earned a Naval commendation from Princeton.

Frances "Betty" Snyder Holberton

They devised sequence diagrams to keep track of what they wanted done, in what order, under what conditions. One of them invented a method of diagramming algorithms using variously shaped boxes and arrows. Her system of rectangles, diamonds, ovals and lines was so perfectly transparent in its flexibility and functionality, it's used to this day: the now-classic flowchart, with its familiar iconography.

The flowchart was just one of Frances "Betty" Snyder Holberton's many inventions. In college, a math professor had bullied Betty into switching out of Math and into Journalism, but she would turn out to become the most accomplished of all the ENIAC programmers.

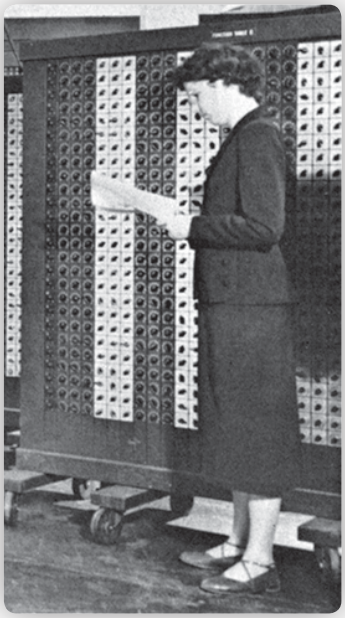
She conceived of the notion of using something like a typewriter to provide data to the computer, and invented the first keyboard input system. She also invented the first statistical analysis system for the U.S. Census. With John Mauchly, she developed the first programming language, the C-10, for the BINAC computer introduced in 1952. For her seminal work in inventing the first sorting routine, the Institute of Electrical and Electronic Engineers gave her its IEEE Computer Pioneer Award. In 1997, the Association of Women in Computing bestowed on her its highest honor, the Augusta Ada Lovelace Award.

Kay McNulty, Frances Bilas, Ruth Lichterman, and Marlyn Wescoff

ENIAC programmer Katherine "Kay" Rita McNulty continued programming also, eventually marrying John Mauchly and teaming up in continuing professional endeavors. You can see Kay's own telling of her ENIAC experience in *a 1977 interview posted on YouTube*[13]. Her college friend

13 *a 1977 interview posted on YouTube:* https://youtu.be/9Jh5SCM75Xg

Betty Holberton



Frances Bilas Spence also stayed in the field. Ruth Lichterman Teitelbaum trained ENIAC programmers for two years. Marlyn Wescoff Meltzer married and left professional life shortly after the war ended. She's one of the interview subjects in the *Top Secret Rosies* documentary.

In 1997, all the ENIAC programmers were inducted into the Women in Technology International Hall of Fame.

The six ENIAC programmers were not the only women who shaped the field of computing since the earliest days. During World War II, the other branches of the U.S. had groundbreaking projects. The Navy's Mark I programming team included *Grace Murray Hopper*[14] and *Ruth A. Brendel Noller*[15]. The Air Force had its own project, *Whirlwind*[16], on which Judith Clapp and her team implemented *real-time control and time-shared computing*[17] for the first time.

Solving the Riddle of the Photos

Meanwhile, I still had a question. What was up with the photos, the ones with all the wire plugging and dial twisting? After all, the ENIAC programmers did nearly all their work on paper—their elaborate "pedaling sheets," as they called them, sequence diagrams, flowcharts, and equations. After they'd worked everything out, they would physically configure the

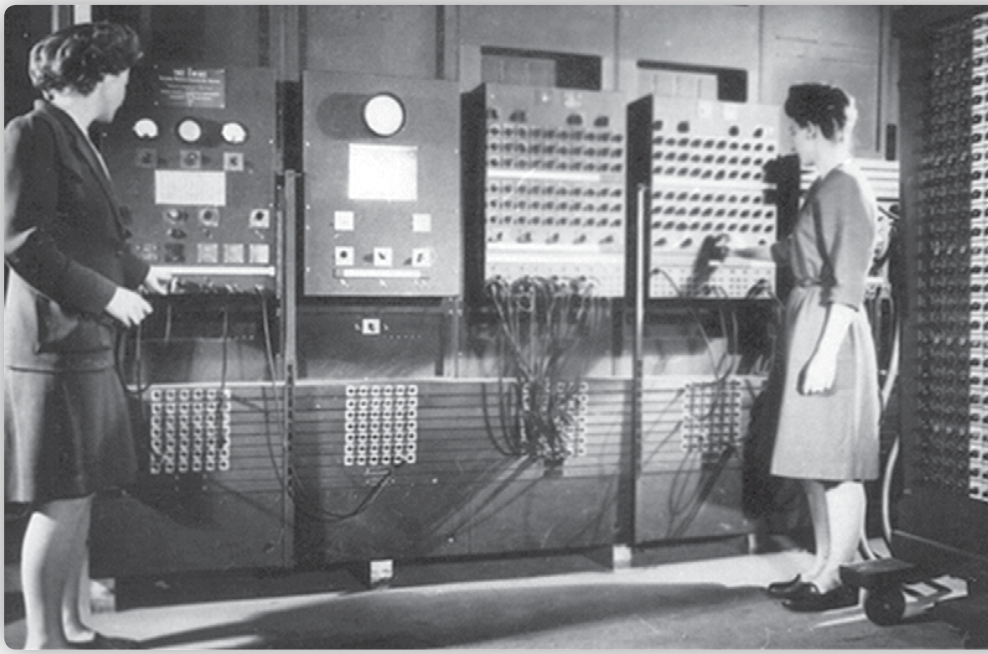
14 *Grace Murray Hopper:* http://phpa.me/hopper-241

15 *Ruth A. Brendel Noller:* https://www.amazon.com/dp/1557509611

16 *Whirlwind:* http://phpa.me/airforce-whirlwind

17 *real-time control and time-shared computing:* http://phpa.me/pioneering-women-cs-pdf

Betty Jean Jennings (left) and Fran Bilas (right) operate ENIAC, U.S. Army Photo



ENIAC. Once it was set up, it could run the program they'd designed. The women didn't keep moving wires around, as the photos suggested.

Jean Jennings explains in her autobiography, *Pioneer Programmer: Jean Jennings Bartik and the Computer That Changed the World*[18], completed just months before her death in 2011.

After ENIAC was presented to the world in February 1946, photographers and movie newsreel makers flooded the lab, along with scientists and dignitaries. Everyone wanted to see the "giant brain" at work. Everyone had questions for the inventors and any other men involved with the project. But no one asked the programmers any questions. "None of them acted as if we women knew anything," she writes. "[They] posed us like models around the computer pretending to perform tasks. We were treated like part of the scenery, doing routine tasks that anyone would do, similar to the 'refrigerator girls' of that era—the models who demonstrated the latest line of home appliances at home expos."

So now I knew. Even while those photos were being taken, the myth was being made: whatever is important about computers is something men do. The visitors to ENIAC could see the big machine, so they wanted to talk to the men who built it. It would be years before the general population came to understand programs and software even existed. Once computing became something done mostly by men; no one realized it had ever been any other way. Except, that is, for the women—and the most enlightened of the men—who were there.

As people became more aware of programming and how important it was, the job rose in status, and it became harder for women to get opportunities to do it. You can find this unfortunate dynamic documented in books including:

- *Gender Codes: Why Women Are Leaving Computing*[19]
- *Recoding Gender: Women's Changing Participation in Computing*[20]
- *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*[21]
- Jennifer Light's article *When Computers Were Women*, reprinted in *Gender and Technology: A Reader*.

Betty "Jean" Jennings Bartik

The most poignant story may be that of Jean Jennings. After several years working at the very center of computer development, she was let go when her husband got a job at the company where she worked. Their policy prohibited employing married couples. Her husband, who soon turned abusive, wouldn't let her work elsewhere. In 1967, after raising their three children, she divorced him. Thanks to her experience, industry connections and great reputation, Jean easily got work in computing. But as years went by, she found herself along with other women, sidelined into less satisfying work. Finally, in the mid-1980s, she couldn't get any job in the field.

Readers of last month's installment will recall this was exactly the period in history where the proportion of women in computing began its permanent (so far) slide from around one-third to around 17 percent.

The story of a hardworking, brilliant woman who watches less experienced men get promoted while she treads water and eventually gets laid off is, sadly, a cliché. In Jean's case, she had actually co-invented the very field she was edged out of. She spent the rest of her working life selling real estate.

Human Computers: Nicole-Reine Lepaute

Nicole-Reine Lepaute was an 18th century astronomer who was on "the first computer team"—a trio of human computers who used the shiny new technology of calculus to predict the 1759 return of Halley's Comet. They weren't just coming up with a date. The most important thing she and her two colleagues had in mind was proving Halley's Comet was a thing. Edmund Halley had the bold idea 76 years earlier that comets orbit and this comet would be back. Nicole's trio worked long days for several months on what was essentially a proof of concept of science itself.

The three friends bucked thousands of years of human tradition and, instead of arguing about ideals of the harmony of the universe, they used known data from previous observations, applied Newton's newfangled laws of gravitation to figure out how much the sun and the planets would be pulling on the comet if it indeed was orbiting through our solar system, and did a whole bunch of math to determine where and when the comet *should* be if the hypothesis *were* correct. Lo and behold, the comet appeared, very close to schedule. (The math and physics work was right on, but they couldn't know there was some data missing; there were more planets out there influencing Halley's trajectory.)

For these and other contributions to astronomy, scientists have named an asteroid and a lunar crater after her. But in *When Computers Were Human*[22], writer David Alan Grier notes another legacy: Lepaute was one of the founders of professional computing. For the next two hundred years, until the advent of the electronic computer, humans worked in teams, systematically dividing work to calculate trajectories of heavenly bodies and ballistics. In doing so, they advanced the fields of astronomy, military science, and industry. The weighty books they filled with their hard-earned computational results

18 *Pioneer Programmer: Jean Jennings Bartik and the Computer That Changed the World:* http://phpa.me/truman-eniac

19 *Gender Codes: Why Women Are Leaving Computing:* http://phpa.me/wiley-gender-codes

20 *Recoding Gender: Women's Changing Participation in Computing:* https://mitpress.mit.edu/books/recoding-gender

21 *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise:* http://thecomputerboys.com/?page_id=20

22 *When Computers Were Human:* http://phpa.me/when-computers-human

were used for years, sometimes even for over a century.

The Analytical Engine

Not only were these results slow to come by—it could take hours or even days of work to compute a single equation—but they were also error-prone. In 1821, a 30-year-old British mathematician named Charles Babbage had a life-changing epiphany while helping a friend slog through errors in new calculations for the British Nautical Almanac. (Side note: English computer Mary Edwards contributed about half the almanac's calculations from 1773 to 1811.)

"I wish to God these calculations had been executed by steam!"[23] he shouted, vexed. Then he realized: Why *not* create a steam-powered computing machine? After all, steam-powered inventions of brass and wood and iron were suddenly everywhere—machines which performed mighty tasks with speed and accuracy like never before. For the next fifty years until his death in 1871, Babbage's life's work became *the design of the Analytical Engine*[24]. If it had been built, it would have been the first general-purpose computer, fundamentally equivalent to the laptop I'm using to write this article, despite being powered by steam, built of wood and brass, and of course, much, much slower. Today we would call such a device "Turing complete."

Mary Somerville

Though she didn't work directly on the Analytical Engine, Mary Somerville played an important role in the project. In the 1830s, the self-taught mathematician and astronomer was making a splash with her beautifully written, breathtakingly comprehensive books on science. Highly technical, yet lucid and accessible, *On the Connexion of the Physical Sciences*[25] wove astronomy, biology, electricity, electromagnetism, geology, light, and sound. It was the Cosmos of its day, and Somerville was the Carl Sagan—and the Ann Druyan, too. (The distinctively lyrical writing in the two Cosmos series is the hallmark of *Druyan*[26], co-writer of the 1980 series and solo writer of the 2014 series Neil DeGrasse Tyson hosted.)

The eminent William Whewell wrote *a review of On the Connexion in the Quarterly Review*[27] in which he suggested a new word to describe someone with Somerville's holistic mastery of the physical sciences: "scientist."

Gregarious and kindhearted, Somerville loved spending time with other scientifically minded women and encouraging their efforts. She was also great friends with the blustering, but generally good-natured, Charles Babbage, who by this time had earned the Lucasian Professor of Mathematics at Cambridge—the same prestigious position held by Isaac Newton and Stephen Hawking.

Augusta Ada Byron King, Countess of Lovelace

Following the custom of the day, in her 18th summer Augusta Ada Byron left childhood behind and "came out" as a young lady in society. Londoners had anticipated this particular girl's first season of balls and parties for years. They would finally get to meet the only legitimate child of the late superstar poet Lord Byron, who had been gorgeous, sexually ambiguous, even incestuous, and altogether "*mad, bad, and dangerous to know*[28]."

Ada's mother, the social activist and mathematician Annabella Milbanke, had done her best to train out any potential "poetical madness." But, despite starting her math lessons at age four and administering them while the child was tied to a board, despite the math teacher who promised his curriculum would *cure the adolescent girl*[29] of imagination, Ada remained untamed. She dreamed of creating "*a poetical science*[30]."

Enter Mary Somerville, who befriended Ada—and who happened to have a standing invitation to one of the most coveted events in London society: Babbage's magnificent parties. She took Ada along, and the experience changed her life forever.

It wasn't splendid surroundings that made this absent-minded professor's shindigs legendary: it was the array of guests so brilliant, so notable, we still know many of their names today, from Charles Dickens to Charles Darwin.

But what left Ada thunderstruck was the work-in-progress piece of computing machinery on display. Here was the avenue for her "*insatiable & restless energy*[31]," the way to fulfill her potential to become a "discoverer of the hidden realities of nature."

Babbage envisioned his Analytical Engine as a way to speed up computation, a way to fill up those books full of tables faster and more accurately. Ada understood a general computation machine could harness the algorithm in a way that gave rise to a new and separate discipline, what today we call computer science. "The science of operations, as derived from mathematics more especially, is a science of itself, and has its own abstract truth and value," she wrote.

Ada, soon to become Countess of Lovelace, worked closely with Babbage for years, until her death from cancer at just 36. His design and her vision matured over time. With the encouragement of both Babbage and Somerville, Lovelace published what became a *classic text*[32] that *influenced*

23 *"I wish to God these calculations had been executed by steam!":* http://www.computerhistory.org/babbage/overview/

24 *the design of the Analytical Engine:* http://phpa.me/anlytic-engine-works

25 *On the Connexion of the Physical Sciences:* http://phpa.me/archive-connexion-physical-sciences

26 *Druyan:* http://phpa.me/star-talk-druyan

27 *a review of On the Connexion in the Quarterly Review:* http://phpa.me/quarterly-review-connexion

28 *mad, bad, and dangerous to know:* http://phpa.me/nytimes-byron

29 *cure the adolescent girl:* http://phpa.me/lovelace-poetical-science

30 *a poetical science:* http://phpa.me/poetical-science-ebook

31 *insatiable & restless energy:* http://blog.stephenwolfram.com/?p=10287

32 *classic text:* https://www.fourmilab.ch/babbage/sketch.html

Diagram for the computation of Bernoulli numbers

By Ada Lovelace, Public domain, via Wikimedia Commons

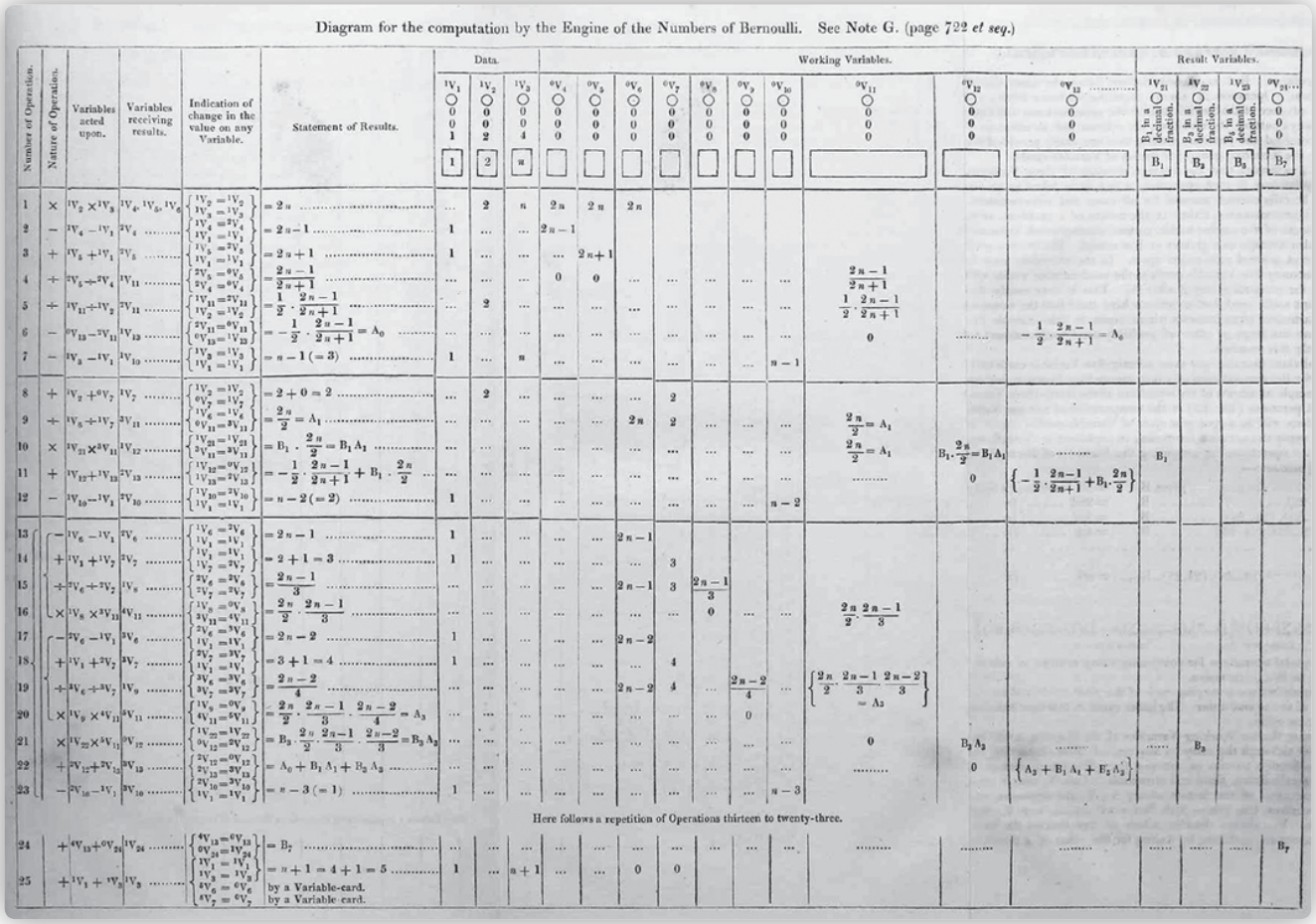


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Alan Turing[33] nearly a century later. Starting by translating of Luigi Menabrea's paper based on talks Babbage gave in Italy, Lovelace added some notes with her own insights. And kept adding, and adding, until she had written several times more than the original paper. She diagrammed, in great detail, what many people call *the first computer program* —though some say it doesn't count because the computer to run it on wasn't built—a routine to calculate numbers in the Bernoulli sequence.

But Lovelace's greatest legacy was being the first to understand, and express, the way in which computers can model systems. It's not just about the numbers. It's about mapping algorithms to the activities of our life. That's how it is computers have so utterly transformed the way we play games, communicate, make music, navigate, heal. They can "weave algebraic patterns," to use Lovelace's much-quoted phrase.

Because of assorted vagaries of circumstance, personality clashes and quirks, ill-timed funding shortfalls, and of course, the early death of Lovelace herself, the Information Age did not start in the 1840s. But it's important to realize it might have. The vision was there. The technology was there. Consider that as soon as the people and the materials began to come together one hundred years later, things happened very, very fast.

33 *influenced Alan Turing:* http://phpa.me/lovelace-turing

Conclusion

When I set out to learn what contributions women had made to our world of computer tech, I thought I'd find a handful of marginally relevant anecdotes. Instead, at every point along the way, I found women and men working side by side, creating worlds of discovery.

There are so many more central figures like Sister Mary Kenneth Keller, Thema Estrin, Melba Roy, Barbara Liskov, Jean Sammet, Roberta Williams—stories I didn't have the space to share here. These are just a few of the many women at the center of revolutions in possibility, history's heroines who have shaped our world.

Sources & Additional Reading:

- *The Oxford Handbook of the History of Mathematics*, http://phpa.me/oxford-history-math
- *Women in Early British and Irish Astronomy*, http://phpa.me/women-british-astro
- *The Thrilling Adventures of Lovelace and Babbage*, https://www.amazon.com/dp/0307908275

Vesna Vuynovich Kovach is part of the small, passionate, agile development team at OfficeSupply.com. She runs DevDivas.com, a site that celebrates the history of women in technology, and tweets at @dev_divas.

Perpetual Process ~~Improvement~~ Perfectionization

David Stockton

As software developers, we should strive to be constantly learning, continually improving ourselves, and our craft. We do this to improve how we write code and how we debug and fix problems. But, we should also be continually evaluating and refining the processes we follow. Rather than blindly following the same process because of someone, somewhere, at some point in the past put it in place, we should be looking for ways to improve how we do things.

In this issue, I want to take a look at the progression of the processes we use at my job, and how we made, or are in the process of, making the transition between the old way and the new way. Now that's dedication[1]!

The Way of Things at Work

At my company we have three teams, each working on a separate software as a service product. Each team runs in a slightly different way, based on the comfort level of the team, the maturity of the software, and the comfort level of the customer in dealing with deployments.

We have one team which has the oldest and most tendril-y code, and their customer base is the least comfortable with deployments as historically, they have the highest chance of going poorly. Currently, they follow a scrum process including planning meetings, backlog grooming, and follow two-week iterations. There's a code freeze period of 3-5 days so the entire system, or as much as is feasible, can be regression tested and UAT'd (user acceptance tested) before deployment to production.

1 *This is a bit of an in-joke at work. We had a résumé come in, listing an item of experience as "We went from the old way to the new way in less than a month. Now that's dedication!" There was no other detail about what the old way or new way was, but apparently going from the old way to the new way in less than a month is proof of dedication.*

> *Scrum[2] is an agile approach where a key principle is that customers can change their minds, also called requirements volatility. A scrum team's focus is to deliver quickly and respond to emerging requirements.*

The second team is following more of a "scrumban" process where they have a prioritized backlog they work through. They utilize on-demand planning meetings as needed, to flesh out stories and find the answers to questions which are not clear. They run a one-week iteration with a short code freeze period.

> *Scrumban[3] is a hybrid allowing Scrum teams to explore Kanban concepts. Primarily, Scrum is used as the way of working while Kanban is used to visualize, understand, and improve the work process.*

The third team doesn't completely follow scrum or kanban, but the process would be closest to kanban if choosing between the two. They report on what they worked on and will be working on, but stories flow through from concept to code continually. Once the code has been completed, it's pushed up as a pull request for review. Once two team members have approved, and the automated build has passed, the code is automatically merged into master and deployed to the QA environment. Another series of automated tests is executed and if they pass the code is promoted to the demo and production environments, again, automatically. This group can deploy potentially dozens of times per day with no downtime.

> *Kanban[4] provides a visual process management where developers "pull" work when they have capacity to take on a task. Kanban encourages limiting work-in-progress and multitasking.*

Bug Tracking at the Beginning

None of these teams got to these points for free. When I started, there was only one team. They were using a bug tracking package called *Flyspray*[5], but doing it poorly. Flyspray wasn't effectively serving its intended purpose. It appears it may be better now, but I cannot speak to that. When I first heard of it, it was a 4th or 5th page Google result for "Flyspray bug tracking," so how the team found it and decided to use it is beyond me.

2 *Scrum: https://en.wikipedia.org/wiki/Scrum_(software_development)*

3 *Scrumban: https://en.wikipedia.org/wiki/Scrumban*

4 *Kanban: https://en.wikipedia.org/wiki/Kanban_(development)*

5 *Flyspray: http://www.flyspray.org*

At the time, everything that was tracked, from feature requests and improvements, to actual bugs and crash reports, were entered into the system as a "bug" even though the system allowed other designations. I feel there is a psychological difference in how a developer (and a product owner) looks at an issue labeled as a "bug" compared to "improvement" or "feature." This was one of the first things I pushed to change. If the request was about broken functionality, it could be labeled as a bug. If it was to implement previously unimplemented functionality, then it was a feature. I hounded our project managers on this and told developers to push back on issues which were labeled incorrectly.

Developers view a bug as something which is messed up or broken in the system.

Developers view a bug as something which is messed up or broken in the system. It's typically a deficiency in the development of the code which leads back to the developer (not all, but many) seeing it as a failure on their part which they could have or should have addressed. When new functionality is labeled as a bug, some developers start to wonder how they are expected to anticipate these shortcomings, which leads to lower team morale since there's no way to anticipate fully everything the product owners or customers might ask for.

The next problem we had was the bug tracking system was "losing" issues. It was the Hotel California where bugs went in and then could never be found again. During this time, the team wasn't following any sort of "Agile" methodology. It was strictly recording "bugs," fixing "bugs," and deploying fixes for "bugs." Then, soon thereafter, deploying hotfixes to fix the fixes. Deployments would often last late into the night or early morning with all hands on deck to ensure when there was a problem, the team could slap together a Band-Aid to patch it and keep things running. At least until they woke up again and found there was another fire to put out.

Making Deployments Less Stressful

These deployments typically took four hours or more, often significantly longer. They would usually be performed with developers and QA onsite, meaning developers and testers would have to run out and grab food before settling into the office for a significant portion of the evening. I started buying the team dinner of their choice to make some aspect of it less awful. The real problem still needed to be addressed. Deployments were long, too error prone, too stressful, and there was not a good reason for it.

We focused on improving process more, to reduce this time. One early change we made was to have the team participate in "Agile" training and we switched from using Flyspray to RallyDev[6]. This brought about workflows and ideas which were not present before. Around this time, the second project team was formed, and work began on implementing their new system. With this team, we put a strong emphasis on automated testing. TDD was encouraged, but also not required. Still, a large number of tests were written.

A short while later, we implemented Jenkins[7] as a continuous deployment server to run these tests. We also switched our source code repositories to Mercurial from Subversion. Mercurial provided a lot of functionality lacking in SVN. It allowed us to start implementing feature branches and merges which weren't going to cause problems when brought together with other code. With the previous SVN repo, it was fairly common to have a developer clobber someone else's changes by committing to the mainline without bothering to ensure someone else hadn't checked in work in the meantime. One particular developer was notorious for hammering his code over everything which had been done in the time since he started his work.

Bugtracking Evolution

A bit later we discovered the way we'd implemented our own Agile processes and how RallyDev wanted you to work, were often at odds with one another which lead to the feeling of fighting with the issue tracking system. Because of numerous workarounds and compromises and the general feeling of, "something's still not quite right," we decided we needed to replace our issue tracker yet again.

We looked at some new alternatives with FogBugz[8] and JIRA[9] being the top two contenders. The FogBugz philosophy on issue tracking is to get out of the way as much as possible. There are very few required fields to get a new issue in the system, and new issues can even come in via an email. This makes it easy to track features and bugs from internal or external customers. When developers work on tickets, they must start by making a time estimate. Then they tell the system when they start work and when they are done. This allows FogBugz to track the accuracy of each developer's estimations individually. It can then use these historical estimations and actual times to run simulations to determine what a developer is likely to be working on and when they are likely to finish. You don't even need to be accurate in your estimations for this to be extremely valuable.

Since the system simulates completion times based on your past estimations, it gives back a model of when you'll be done. Essentially, it runs a Monte Carlo simulation 100 times and then aggregates those to form a probability model. If you are accurate, then the estimates for when

6 *RallyDev:* https://www.rallydev.com

7 *Jenkins:* https://jenkins.io

8 *FogBugz:* https://www.fogcreek.com/fogbugz/

9 *JIRA:* https://www.atlassian.com/software/jira

you'll likely finish will probably not have too much variation. If you are consistent, but inaccurate, (you often over or underestimate by a common factor) there will probably not be a lot of variation for when the system thinks you're likely to complete a ticket, but it will take into account your estimates and reality don't necessarily align. If you're all over the place with your estimates, then there is likely to be a good deal of variation on when the system will think you're likely to be complete. All in all, I think that sort of project management or evidence-based projections is pretty excellent. Unfortunately, I've not had any luck getting FogBugz into the organizations where I've worked.

Ultimately, we decided to go with JIRA. We imported a lot of the historical tickets from Rally, some of which had been dragged along, kicking and screaming, all the way from Flyspray. JIRA is nice because it allows you to determine nearly every aspect of your workflows, which you can do on a number of different levels. If you want, you can set up different workflows on a per-project basis, or even at a per-issue-type basis. This meant for each of our teams, we could set up processes and workflows in our issue tracker which match the way each team has decided to work. The downside is there's a lot of configuration which needs to happen before this works flawlessly. Fortunately, the provided workflows are a good place to start.

Kai Zen

Evolving our workflows in JIRA has been an ongoing process. Evolving our processes has also been an ongoing process. I want to ensure each team is constantly looking for ways to make things better. Part of this is to make sure we don't feel any part of our process is "sacred." We may find certain patterns or ideas work well, hopefully, better than all the other things we've tried, but even those aspects should be questioned for potential improvements.

Not every experiment will be better, but that's OK too. The Japanese term, *Kai Zen* means improvement or continual improvement, and it's something we do strive for all the time. It means we try not to sit on our laurels or stop trying new things just because where we are and what we are doing is comfortable and working.

For the third team I spoke of, we started out with the idea we'd like to get to a continual deployment process. This didn't come about overnight. When the project started, we had a manual deployment process, which was not on a set schedule. Each bit of code could go through the process to production, but each step along the way there were scripts to run. We eventually got to the point where those scripts were run when you clicked a button in Jenkins.

After a while and many automated tests, we felt more confident in the process so we decided to make the pushing of the deployment button more automatic. Pull requests would be merged at the request of the QA team when they were ready to test a particular feature or bug fix. This worked well for a bit, but as the development team grew and the number of things to test increased, we found waiting for a person to merge code in order to test was a new bottleneck. Most of the code coming from this team included unit tests, ensuring most of it would act right, guaranteed. In reality, because the majority of the happy path and standard bug checks had been automated the majority of the bugs QA was catching were the odd cases; the one-off edge cases that were unlikely to affect a real customer. So we decided to upgrade the process again.

This time, we decided we still felt code review was important and necessary (almost always). We felt the unit tests, as well as system integration and behavioral tests were important. Adherence to the coding styles is also important, as well as keeping API tests working. These became jobs in Jenkins which could run and report back to Bitbucket as well as kicking off additional jobs. At this point, much of our workflow has been automated.

From a developer point of view, we pull a ticket off the top of the backlog and mark it in progress. The code is developed in a new branch off of the master (did I mention we switched from Mercurial to Git along the way as well? Well, we did that too, because it allows for even more flexibility in our workflows). Once the code is done, a pull request is created. This starts automated builds which ensure all the unit tests are passing, and the code adheres to our coding standards. We used to do a lot of other static analysis as well—building PhpDocs, PDepend, and others, but we found we were not using the results of those programs very often, if ever, and several of them add a significant amount of time to the build. So we evolved our process to remove those. We still run them, but they are in another Jenkins job which doesn't affect our "critical path" flow. Essentially, if the PHPUnit, phpspec, the JavaScript unit tests, and phpcs jobs complete successfully, we count that as good. We still get stats from copy paste detection or project mess detection, or code coverage, but those happen off to the side in a separate job.

If everything passes, our webhooks project awaits indications from Bitbucket the code has received the proper number of approvals from the right people. We also put in place a check to ensure all the code has been properly rebased and squashed. If it's not rebased, the system will let the developer know in Slack they should rebase, and gives them a button which will do it for them. It will not squash, though, the developer must do that on their own.

> *Rebasing affords a clean project history by eliminating extraneous merge commits. You do lose some context since new commits are created for your feature branch. Atlassian has a good tutorial explaining rebasing versus merging, Merging vs. Rebasing*[10]

10 *Merging vs. Rebasing:* http://phpa.me/atl-merge-rebase

If all those checks pass, the system merges the code. This starts the next chain of events, deploying to QA automatically, moving the JIRA story from "In Progress" to "In-Review," kicking off API and BDD tests. If those pass, then the code is deployed to the Demo and Production environments. All-in-all, it can take less than 20 minutes to get the code from your brain into production. Similar to the process seen at places like Etsy, new developers on this team can deploy to production on their first day.

All of our teams currently use some version of the approval and build-check => merge process. Each differs in how many approvals must be in place in order to merge, along with other team specific approvals by certain individuals. This has all evolved out of what works per-team and it's set up to be easily changeable. It allows a team which may not yet be ready for a process like "continuous deployment" to work towards it as a goal and get used to the mindset which needs to be in place to make that work without causing problems.

A Word on Continuous Deployment

Personally, I am a big fan of continuous deployment. It allows code to move quickly out to customers where you can receive feedback right away. You can see the impact your changes make immediately rather than days, weeks, or months later. However, it means features and stories, and sometimes even bug fixes may need to be broken out into a series of sequential steps, each of which gets you closer to the complete change, but individually can be done with little or no customer-visible impact on the running system. This takes a lot of discipline to do right. Ideally, stories are small and can be done with no impact on any other stories, but in practice, this is not always the case. Let's take a quick look at an example.

Suppose you need to make a change to the system which allows the database to store time zone information where previously it has been left out. If you're doing a big bang deployment where there must be coordination and downtime, this is pretty simple. First, you make the changes to the code, the API, and the database. Then, you script up something to update the database field to use time zones, and you build a query or a script which will run when you deploy. This may mean there's downtime on the site depending on how long it takes to convert the data.

If you're doing continuous deployment, you don't want this downtime since deployments can happen at any time, not just when the application is less busy. It requires breaking up the tasks into small ordered pieces which can each run without causing problems. At each point, each individual change can be deployed without the site needing downtime. It may be more work overall, but I feel it's worth it. Here's an example of how to approach the same problem with a Continuous Deployment mindset. Keep in mind after each step is done, it would be deployed completely before pulling in and deploying the next step.

1. Add a new time zone column to the database.
2. Script to bulk migrate old non-timezone column to the new one.
3. Triggers or functions to keep writes to old column synced into new column.
4. Change APIs and code to return the new column, leaving the old one in place.
5. Change the UI and other clients to use the new column, removing references to the old column.
6. Change the API to stop returning the old column.
7. Remove the synchronization code and old column from the database.

If these steps are done in this order, then there's no need for any downtime. If some steps take a long time (such as data conversions or the addition of new columns) these can take as long as is needed but the site remains up and running. Since nothing relies on the new database pieces until they are fully deployed, it doesn't create a problem. Because we remove any reference to the old stuff from the top down when it's removed from each level, there's nothing relying on it, so there's no breakage either.

Don't Stop Looking for Ways to Do It Better

Chances are, if you're doing some or all of these, you may feel things are pretty darn good. And they are. The process and flow we have at this company is much better than what I've had at previous companies. But that doesn't mean they cannot be better. I challenge you to continue to look at your processes and find ways that you think might make things better and give them a shot. If it doesn't work, that's OK, stop doing the things that don't work or don't help, and keep doing the things which do work and make things better.

If you are consistent about making these improvements all the time, there's no practical limit to how good things can be. I often hear from former employees that what we have regarding process is the best they've seen anywhere they've worked, both before and since. It's usually at this point I get an opportunity to tell them all the things we've changed since they left and how much better it is now. Keep improving and leveling up. See you next month!

David Stockton is a husband, father and Software Engineer and builds software in Colorado, leading a few teams of software developers. He's a conference speaker and an active proponent of TDD, APIs and elegant PHP. He's on twitter as @dstockto, YouTube at http://youtube.com/dstockto, and can be reached by email at levelingup@davidstockton.com.

Getting the Most out of a Conference

Cal Evans

If you have never been to a PHP conference, BUY YOUR TICKET NOW! By the time this comes out the winter conference will have all opened their ticket sales and the spring conferences are right around the corner.

Last month, I wrote about how conference regulars, speakers, and organizers could make conferences friendlier to newcomers. But you don't have to wait for them to listen to me before going to your first one. The problem many first timers have is they don't know what to expect, or what to do. So, let me share with you what I've learned in ten years of being a professional conference attendee.

Know What You Want to Accomplish by Attending

OK, I hear you hollering at me "Cal, **I want to learn**!" Yes, we all do. But WHAT do you want to learn? Is there a particular problem you are trying to solve? Is there a new technique or technology you want to know more about? Even if your boss just handed you a ticket and said, go, you should put some thought into what is important to you and how you can maximize the knowledge you bring home. Look at the technologies you are using already at work, but don't limit yourself to them! If you are interested in learning about functional programming, message queues, or a new framework—that knowledge will be useful months down the road.

Review the Schedule and Select the Must See Sessions

Once you've identified what you want to learn, look at the schedule. Identify the sessions teaching what you want to learn. Identify any secondary sessions you think might be interesting. Spend a little time before you get to the conference figuring out the sessions you want to see. If the conference has an app or mobile site, check it out to see if it'll be useful. Some conferences will let you build a schedule online and send you reminders when the sessions you selected are happening.

Make a List of the People You Want to Meet

Are there speakers you want to meet or discuss a problem with? Make a list. Trust me; two days sounds like a lot of time, but it will go by fast. If you are not organized, time will slip away without you getting done what you want to get done. Speakers are usually friendly and approachable, but make sure you're not trying to meet them right before their talk. They'll appreciate a little time and space to finish preparing and getting mentally prepared to give their presentation.

Do Not Make Plans for Every Session

It is a rookie mistake to plan out every session. Keep yourself flexible. Try to leave one session each morning and every afternoon empty. If you don't fill them, you can always go to a session. However, there will be times when you get involved in a discussion with someone, you meet someone on your list and they are answering your question, or you just need to sit and digest about what you've learned so far. Leave yourself some time to think.

Don't Overdo the After Events

I get it; you paid for the ticket, and you want to squeeze every bit of goodness out of it. You want to do it all. Stop; be smart about this. There is the temptation to "go all night." Especially if you've met new people and made new friends. You want to talk, you want to share ideas, you want to keep the fun going. This feels great—until the next morning. Then, you are firing on six out of your eight cylinders. You aren't feeling your best and you aren't performing at your peak. Be reasonable, and most importantly, if there is alcohol involved, know your limits. Know these limits going into the evening, don't try and figure them out mid-way through. Know when you want to be back in your room so you can get a reasonable amount of sleep, and if you are drinking, know how many you can have before bad decisions start to sound like good ones.

Take and Share Notes

Find a way to take notes that will be useful for you later. You should share these via a blog post, ideally. Your team back home might learn a thing or two from your notes, and just the act of explaining and teaching them what you learn will solidify your understanding. Finally, when it comes time to rate the talks (and speaker's really appreciate honest, constructive feedback) your notes can help you remember the talk and explain how it could be improved.

Make Time for Yourself

Finally, make sure you are having a good time. Don't let the pressure of having to fill every minute, having to meet everyone on your list, or having to attend every after-hours event suck all the life out of the event for you. Part of what I get out of conferences is a chance to get away from it all while still being immersed in tech. I can focus on tech but still take the time to reflect on where I am, where my career is, and where things are going. Make time for self-inspection and reflection. To get the most out of the conference, you should come back informed and energized. It should be a time to recharge your batteries, as much as it is a time to get inspired by new technologies and now tools. And, depending on expectations back at work, you may need some time to answer emails from your boss and colleagues who are covering for you while you learn.

Conferences are great! If you have never been to one, make an effort to get to one next year. Your career will thank you.

Past Events

October

Bulgaria PHP 2016
October 7–9, Sofia, Bulgaria
http://bgphp.org

Brno PHP Conference 2016
October 15, Brno, Czech Republic
https://www.brnophp.cz/conference-2016

ZendCon 2016
October 18–21, Las Vegas, NV
http://www.zendcon.com

International PHP Conference 2016
October 23–27, Munich, Germany
https://phpconference.com/en/

DrupalSouth
October 27–28, Queensland, Australia
https://goldcoast2016.drupal.org.au

Forum PHP 2016
October 27–28, Beffroi de Montrouge, France
http://event.afup.org

ScotlandPHP 2016
October 29, Edinburgh, Scotland
http://conference.scotlandphp.co.uk

Upcoming Events

November

TrueNorthPHP
November 3–5, Toronto, Canada
http://truenorthphp.ca

php[world]
November 14–18, Washington D.C.
https://world.phparch.com

December

SymfonyCon Berlin 2016
December 1–3, Berlin, Germany
http://berlincon2016.symfony.com

ConFoo Vancouver 2016
December 5–7, Vancouver, Canada
https://confoo.ca/en/yvr2016

PHP Conference Brazil 2016
December 7–11, Osasco, Brazil
http://www.phpconference.com.br

January 2017

PHPBenelux Conference 2017
January 27–28, Antwerp, Belgium
https://conference.phpbenelux.eu/2017/

February

SunshinePHP 2017
February 2–4, Miami, Florida
http://sunshinephp.com

PHP UK Conference 2017
February 16–17, London, U.K.
http://phpconference.co.uk

March

ConFoo Montreal 2017
March 8–10, Montreal, Canada
https://confoo.ca/en/yul2017/

April

DrupalCon 2016
April 24–28, Baltimore, MD
https://events.drupal.org/baltimore2017

May

phpDay 2017
May 12–13, Verona, Italy
http://2017.phpday.it

php[tek] 2017
May 24–26, Atlanta, Georgia
https://tek.phparch.com

Photo Credits

Photos in this article used with permission from Eli White's Flickr account[1].

These days, when not working with PHP, Cal can be found working on a variety of projects like Nomad PHP. He speaks at conferences around the world on topics ranging from technical talks to motivational talks for developers @calevans.

1 *Eli White's Flickr account: https://www.flickr.com/photos/eliw/*

Do You Truly Know Your Tool of Choice?

Matthew Setter

IDEs and text editors, could we honestly develop applications without them? You might feel you're some über-geek, who could write code in his sleep and in, the morning when he awoke, he'd have written a better operating system than Tony Stark did for Ironman. But for the rest of us mere mortals, we need the assistance which professional IDEs and text editors give us. In this month's Education Station, I'm going to show you a range of features from some of the leading IDEs and text editors. I hope there'll be at least a couple of features which you haven't tried yet, ones which you can use in your daily work to improve your efficiency and productivity.

Actually, assistance is quite an understatement. Text editors and IDEs save us from searching PHP.net for function names and their parameter lists. They can auto-format code for us, ensuring it adheres to coding standards and conventions. They can autolint code for us, letting us know if we've made mistakes, or attempted to use classes, methods, or objects which don't exist. They can do all of this, and so much more.

When you think about it, while often all we see is a blinking caret staring back at us, they're doing so much behind the scenes so we don't have to. And therein lies the problem.

We often get up and running with a new IDE or editor in a flurry of enthusiasm. We dive in for the first few days and learn a host of new features, settings, and shortcuts. But then the monotony of the job returns. We promise ourselves we'll continue on our journey of discovery, plumbing the depths of what our new tool of choice offers.

But, as with anything in life, how often is that true? We end up using the same, limited, set of features over and over again, never stepping beyond their safe and comfortable confines. As a result, we never know the full power which is on offer.

I hope, in some small way, I can relight the spark of fun, childish excitement, and wonder which perhaps you had when you began using your tool of choice.

We can see, even with a cursory search through any of the online app stores or package managers, there are many tools on offer. But, naturally, in a column of this size, I can only cover so many of them with any sense of justice.

Given that, I'm sticking to four of the best and most widely known tools. These are:

- *Vim*[1]
- *PhpStorm*[2]
- *Atom*[3]
- *Sublime Text*[4]

Have a look at the section on your favorite and see if I've taught you something new. Let's start with my personal favorite, PhpStorm.

PhpStorm

I'd be lying if I said that PhpStorm wasn't my editor of choice. Ever since I was introduced to it about five years ago, I've been hooked. Out of the box, it has a veritable cornucopia of features on offer, and it's so fast. My top three features for it are: *refactoring*, *step-through debugging*, and *code navigation*. It's also cross-platform and runs on Windows, Mac OS X, and Linux flavors as long as you have Java available.

Refactoring

If you've ever worked on a codebase that's a big ball of proverbial mud and you've been able to improve it, you'll appreciate the value of refactoring. Whether it's refactoring code out into separate methods, variables, and constants, changing method signatures, or renaming classes, this is an essential task.

However, if we're not careful when we're doing it, it can result in the creation of bugs. That's why this is my number one feature in PhpStorm. To have PhpStorm refactor code for you, you first highlight the block of code. You then choose the refactor you need done, from the Refactor menu. It will search your entire codebase, even linked codebases, looking in files, strings, and more for the required places to make changes, and then make all the changes for you.

Step-Through Debugging

Ever tried to use `var_dump` to debug a codebase more than a few files in size? Ever tried to debug using `var_dump` on a codebase that has more than a few hundred files? Ever tried to do it quickly, without performing a page reload again, and again, and again—and again?

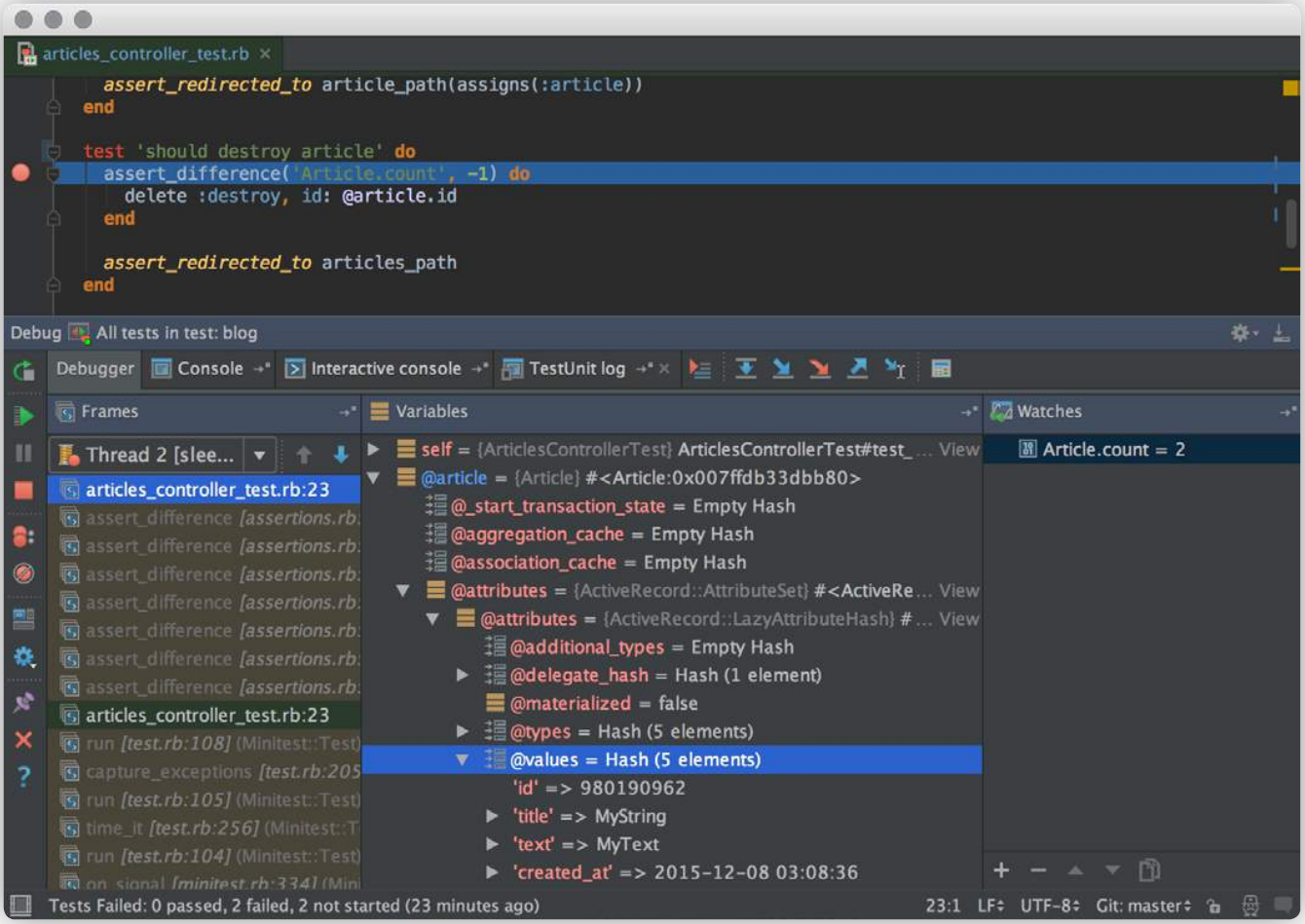
1 *Vim:* http://www.vim.org

2 *PhpStorm:* https://www.jetbrains.com/phpstorm/

3 *Atom:* https://atom.io

4 *Sublime Text:* https://www.sublimetext.com

Figure 1



You may have, but was it all that efficient? I think you invested a lot of time but didn't learn all that much in the process. Instead of wasting your time, use step-through debugging. The first time you do it, it may seem like magic.

After you've done it more than half a dozen times, and try and use `var_dump` again, you'll feel like you've jumped back to the stone age. When setup correctly, all you need to do is set a breakpoint in your code where you want the execution of the request to pause, then make a request to your code. When execution hits the breakpoint, it will stop and provide you with comprehensive information about the application, at that point in time.

What's more, you can step backward, through every step in the call stack, to the initial method call which triggered it off. Take the example in Figure 1, which I borrowed from JetBrains. You can see we've stopped on the execution of an assertion, and there are two variables in the stack, which is the middle of the three columns in the bottom window. You can see their types and values, along with values and properties which they contain. Try doing that with `var_dump`.

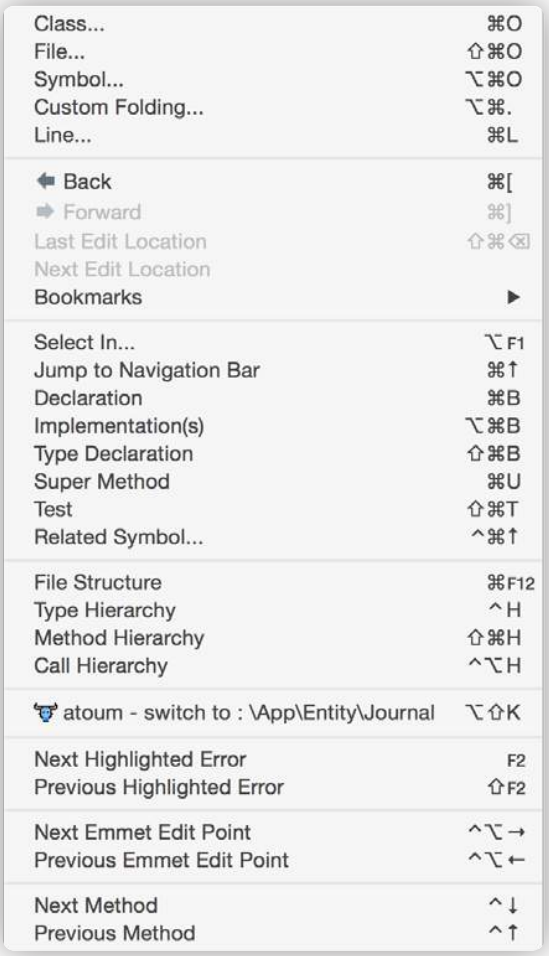
Code Navigation

This one is my third must-have feature—especially on large codebases, or codebases where it's essential I get up to speed quickly. Instead of having to find and open an object's class, I can simply cmd+click, on the Mac, on the use statement and be taken right to it, regardless of where it is.

In addition, I can perform a range of other, powerful, navigations, such as searching anywhere in the codebase, for any symbol, class, or file. I can move to the parent class of the current one, to the parent method of one being overridden, to a test which makes use of the current class or method, or to the declaration of a method, plus oh so much more.

But it's not just symbols, classes, and files where navigation has been thought about. When you have been tracing execution through some files, perhaps as many as 10 or 20, following method calls, and class hierarchies, it's easy to get lost up in the flow of execution. PhpStorm lets you move forward and backward through the locations where you've been working. Have a look in Figure 2, where you can see the full scope of the navigation options on offer.

Figure 2



Vim

Vim, the 30+ year-old veteran, code editor—flame wars have been fought over it. What's more, it's one of the most widely distributed pieces of software in the world. But unlike PhpStorm and other IDEs, it's not as easy, at least for me, to pin down the top features.

I say this because out of the box it does very little. With a little bit of elbow-grease and Google searching, you can configure it to be almost exactly whatever you want it to be. That said, here are my personal picks.

It's Everywhere

That's right, everywhere. There are not many pieces of software which can lay claim to this kind of pervasiveness. Surprisingly, Solitaire beats Vim hands down. But that's a story for another day.

This might not seem like a lot, but if you had to learn only one text editor, then VIM would have to be your top choice. Whether you're working on your local MacBook Pro, Linux desktop, or remote UNIX server, you're going to find a copy of Vim (or at least Vi) installed and ready to go.

To be fair, it's not always the same version, nor configured to the same level of functionality. But, I believe I'm correct in saying if you have learned the Vim basics, then you'll do just fine. It's not the easiest of editors to learn, especially if you're coming from Windows.

Vim is hands down the best choice when you can use it locally, not have to install remote software, configure an RDP or another form of remote desktop session, and use an editor which is quite light on resources, just to do some minor code checks or server configuration changes.

Extreme Customizability

This one has to be on the list of must-knows for Vim. As I said at the start of this section, Vim, as it comes installed is a very limited text editor. However, with the right set of plugins and configuration options it can become, proverbial, putty in your hands.

For example, I do a lot of writing using Vim as my editor of choice, and to make it better suited to the task, I've installed and configured a range of plugins. To quote the Grumpy Programmer himself, Chris Hartjes:

> *I feel very strongly that, any editor that does not allow you to extend through plugins that you can make yourself should not be used.*

To a large extent, I have to agree. While I don't do a lot of customization, the ability to tweak and tune it to what I need has made Vim my must have tool of choice for writing. I'd be nowhere near as productive without it. That said, here's a look at some of the plugins which I use on a regular basis.

vim-surround

Vim-surround[5] makes it simpler to surround a given piece of text with something else. This can be wrapping a line of text in a tag, such as an `<li>` or a list of `<li>` tags in a `<ul>`, or something more sophisticated.

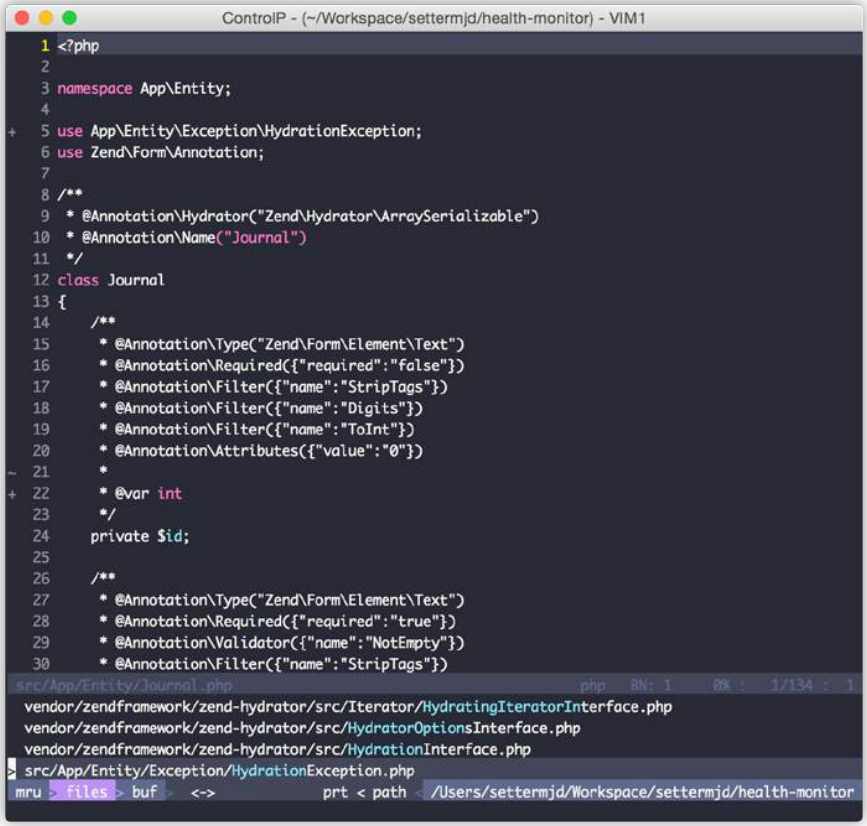
CtrlP

CtrlP[6] is an excellent tool for quickly finding the file(s) you need. You don't have to know their names exactly. Just press ctrl+p and start typing something in the name of the file. CtrlP will then do its best to create a list of files matching your search criteria. You can see an example of this in Figure 3, where I'm searching for the HydrationException class in a personal project of mine.

5 *Vim-surround:* https://github.com/tpope/vim-surround

6 *CtrlP:* https://github.com/kien/ctrlp.vim

Figure 3



vim-airline

Vim-airline[7] is a very funky, very astute, and even very aesthetically pleasing plugin which customizes the status bar (or tabline) when working in Vim. It can provide you instant knowledge of a range of things, such as the name, size, and type of the current file, the current git branch, the encoding type of the current file, the current line and column number and more.

UltiSnips

UltiSnips[8] is an excellent plugin, one I learned from Vim guru, Drew Neil[9]. With it, you can set up a whole series to create a series of expandable snippets. You may be familiar with other editors, such as PhpStorm, which when you type `pubf` and then click `<tab>` or hit `<space>` it will expand it into the following snippet:

```
public function () {

}
```

UltiSnips gives you all that power, plus perhaps more in Vim. It's a must have!

vim-pencil

As the name may imply, vim-pencil[10] customizes Vim for writing as other plugins customize it for coding. With it, you can do the following and a whole lot more:

- Agnostic on soft line wrap versus hard line breaks, supporting both
- Auto-detects wrap mode via modeline and sampling
- Adjusts navigation key mappings to suit the wrap mode

The Dot Rule

One of the best aspects about Vim (any flavor, not just the command line) is you can, in normal mode, use the dot to repeat the last operation. This might not seem like a lot, but take the following as an example of just how handy it can be. Say, for example, I want to find the text `"Roger Rabbit"` in a large script file I'm writing for a remake of the late 80's film *Who Framed Roger Rabbit*. Yes, I'm that old!

You could then use the command `/Roger Rabbit` which would find (and potentially highlight all matches) taking you to the first match. If there were a number of them, it would be annoying to have to re-type the command repeatedly. Instead, with the dot you can just press it to repeat the last action, in this case, a search.

Now how about one a little less trivial? Let's say I had the text in Listing 1. You can see that it's an HTML document with no documentation. I could indent the first list by visually selecting all the list items, then using `^+>` (which translates as shift and right angle bracket). If that wasn't far enough in, I could click the period again to repeat it.

LISTING 1

```
01. <div class="jumbotron">
02. <h1>Health Monitor</h1>
03. <p></p>
04. </div>
05.
06. <div class="row">
07. <div class="col-md-12">
08. </div>
09. </div>
```

Then, I could go down to the first list item in the next list and click period and it would indent the entire list for me. Now here's something a bit more fun. As we determined the first time we indented, the indent didn't go far enough. Instead of using the dot twice, we can instead click `2.`, which will repeat the last action twice.

7 *Vim-airline:* https://github.com/vim-airline/vim-airline

8 *UltiSnips:* https://github.com/SirVer/ultisnips

9 *Drew Neil:* http://drewneil.com

10 *vim-pencil:* https://github.com/reedes/vim-pencil

Atom

Here's an editor which arrived to mass acclaim. I admit, for that very reason, that I initially ignored it. I'm not the kind of person who jumps on the bandwagon, just because "everyone else" is. Herd mentality is usually a good reason to step back and hold a wait-and-see type approach.

Then, if the momentum continues, take a look and see what all the fuss is about. Invariably, if the momentum survives, then more than likely there is something of great value which is driving the uptake.

That's exactly how it was with Atom. I sat back, waited to see if the buzz was going to die down; but as yet, it hasn't happened. So I took Atom out for a spin. Here are the three best features I found.

Customizability

This isn't a characteristic unique to Atom. Any other editor or IDE in this list has it. But Atom seems to take it to a whole new level. It's built with JavaScript, CSS, and Node.js. So it's effectively a web app in another skin if you will. What's more, Atom self-describes as a "Hackable Text Editor." It's no surprise you can customize almost anything, whether that's the UI, theme, or existing functionality. If you need to, invariably you can.

Language-Specific Setup

Again, this isn't unique to Atom alone. Taking PHP as the language of choice, PhpStorm does it out of the box. That said, Atom does a particularly excellent job of it. I've never found that it works just right with PHP. When working with other languages, specifically Go however, it's a dream.

When I say it never worked quite right, what I mean is when I'm seeking context-specific suggestions for function names, class names, and other symbols, the suggestions were a broad mishmash of anything and everything which might, potentially fit.

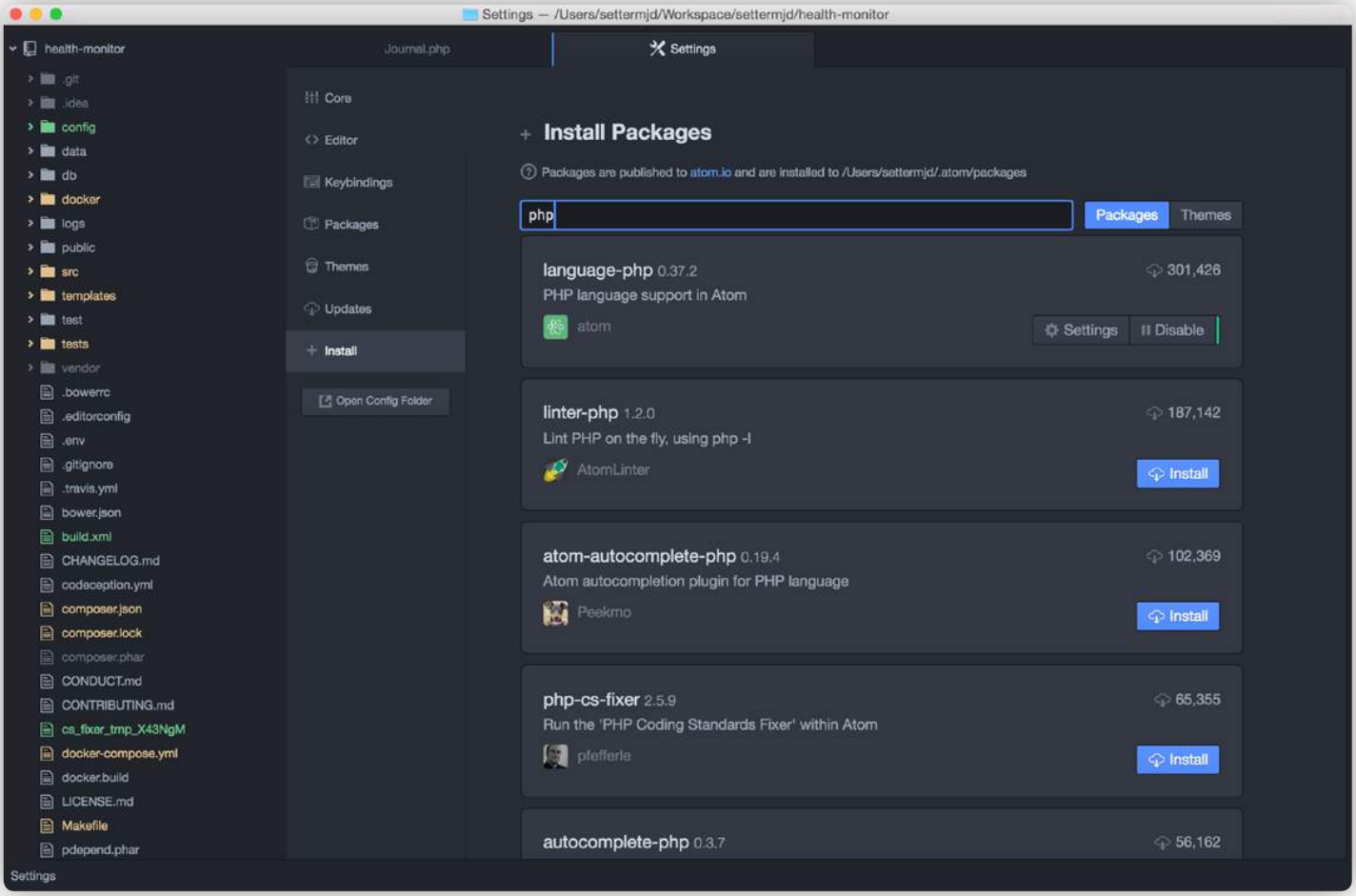
Given that, I've found attempting to emulate intelli-sense style auto-completion with PHP to still be quite troublesome. However, for code linting, code formatting, syntax highlighting, and a whole host more, Atom does an excellent job. It always feels light and nimble.

Packages for Anything and Everything

This is something any modern editor or IDE needs if it wants to shine. If it doesn't have extensibility, and a community of people willing to create said extensibility, the shelf life won't be long. People always have a vast array of different needs, brought about by their different circumstances.

As I write this, Atom's package repository has almost 5,000 packages available, some of which you can see in Figure 4.

Figure 4



They cover everything from beautifiers to linters, from lexers to themes. Even if you're just starting out with Atom, there's more than likely enough packages which you can combine to create just the right environment for what you need to do.

I found that attempting to add in everything and the kitchen-sink doesn't always work out so well. I've found it best to only use what you need and take it one step at a time. Your mileage may vary.

Sublime Text

Sublime Text, often referred to simply as "Sublime" is self-described as: "The text editor you'll fall in love with." Sublime does an excellent job of being an accomplished text editor for developers and, perhaps, designers alike.

Regardless of the operating system you're developing on Sublime is available for it. What's more, it's very light and very responsive. It's no surprise it has taken off with developers looking for an alternative to the often heavy-weight, and laggy, IDEs, such as Eclipse.

While I don't use it near as actively as Vim or PhpStorm, it's still a text editor I turn to from time to time. It's also one many of my colleagues often cite as their first port of call; regardless of whether they're developing in Ruby, Python, Java, or Go. Here's a look at three of its best features.

Navigate Anywhere

Like I said for PhpStorm, being able to navigate anywhere within a codebase, just by looking for something about the item you want to find is an indispensable feature to have. As easy to use as PhpStorm's, you click cmd+p on the Mac, and it pops up a text input, and you can type in anything relating to what you're trying to find.

Sometimes, let's be fair, you're not even sure exactly what you're after. So starting from some arbitrary point, and narrowing in can be a definite blessing. Take the example in Figure 5, where I've opened the search box, and searched for 'Jour'. You can see that it's highlighted, a lot like Google does, the aspect which matched, and provided the full path. I don't even need to pick an item from the list if it is a close enough match. It will automatically open for me.

Multiple Selections

This is a feature I find really wicked, if for no other reason than the way it looks when I'm doing it. Ok, so it's handy too. You can do the same thing in Vim or any IDE by running a global search and replace. Let's be fair. But, there is something about this approach that's so much simpler, and again—visually cool!

Figure 5

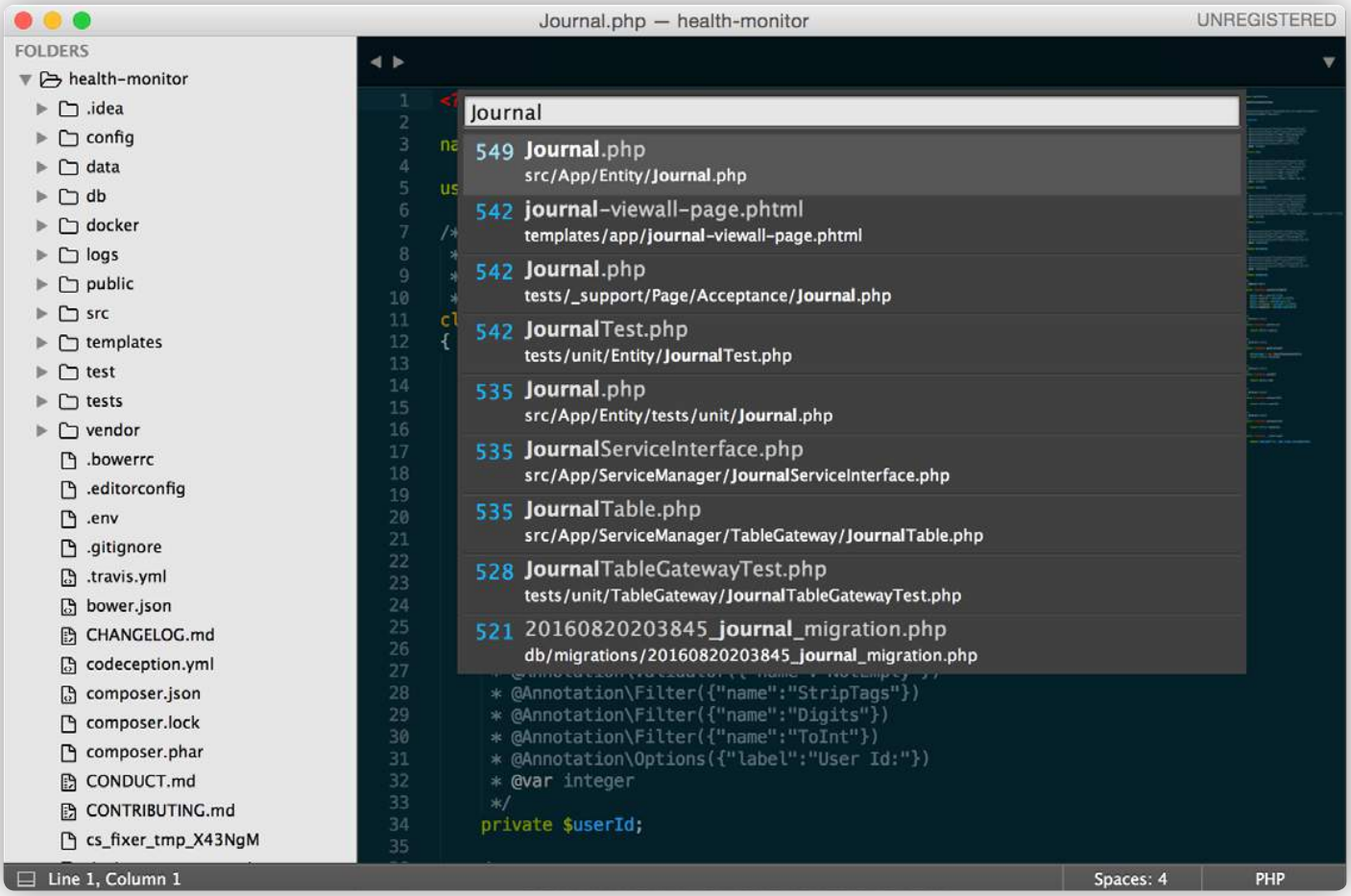
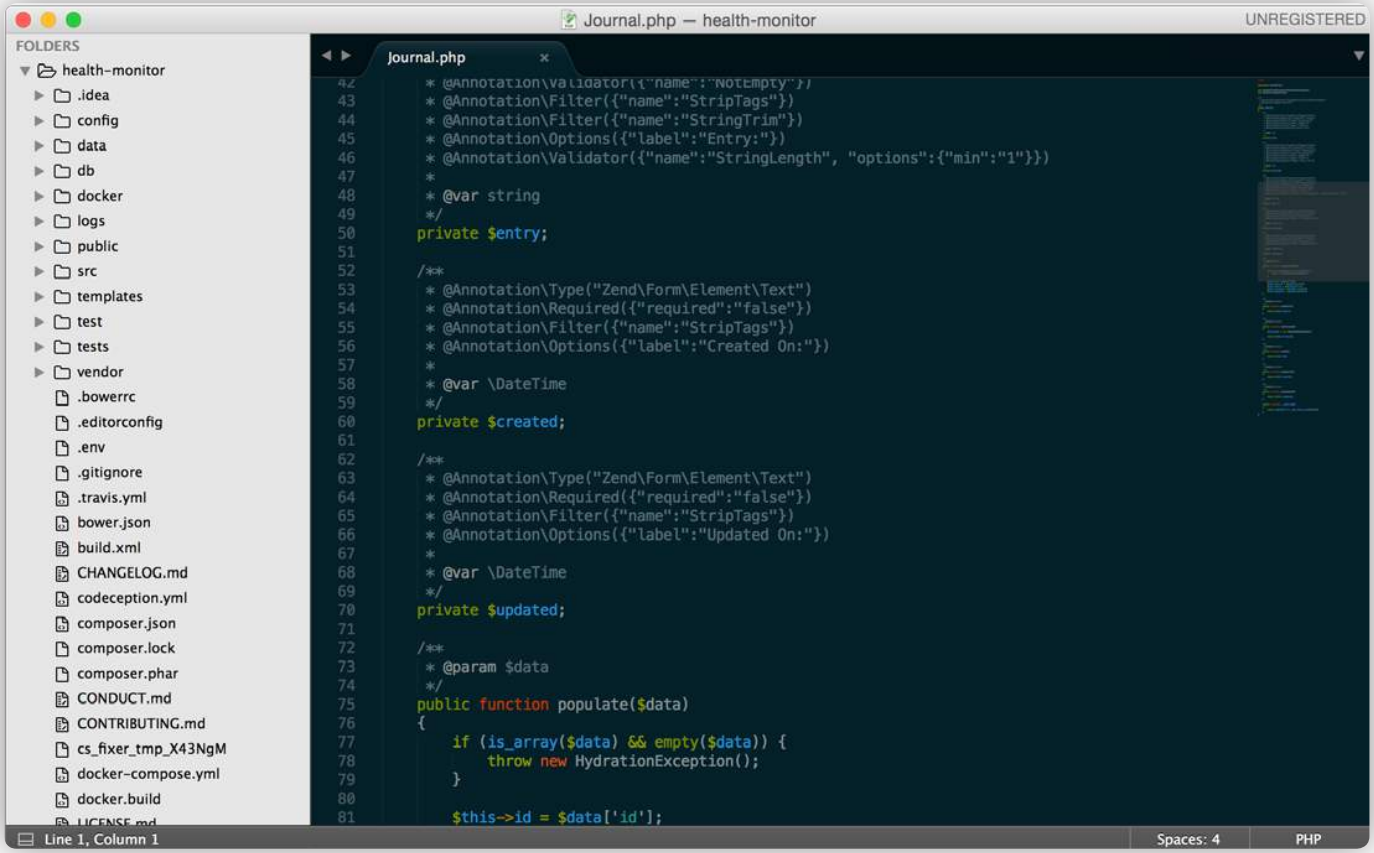


Figure 6



Say, for example, I'd made a mistake in the screenshot below. Instead of having `@Annotation\Filter`, it should have been `@Annotation\Filler` instead. All I'd need to do to change every occurrence is to select one of the incorrect instances, then use ^ + ⌘ + G on Mac, or ALT + F3 on Windows or Linux.

This would then select and highlight all instances of that text in the current file. Then, I just need to change the text and watch as Sublime, in all its glory, changes all the occurrences at the same time. It's nothing if not impressive.

The Minimap

I think this feature is unique to Sublime Text. By default, when you open a file in Sublime, it will feature a mini version of the entire file on the right-hand side, which you can see in Figure 6. It isn't possible to read the information contained in it, but it gives you a good idea of just how long the file is, as well as allows you to navigate throughout the file.

Is this a really compelling feature, or is it more of a novelty? I flit between how I see it quite often. Despite that, I've always kept it around. Something about having it there makes it simpler to work with files.

In Conclusion

Did I cover a feature in your favorite text editor or IDE, which you weren't aware of? I hope so. If I have, then give it a go! Try it out and see if it helps you become more efficient and more effective than you already are. If you're in doubt, try it out anyway. The gray matter always needs a workout. Plus, you never know when it might come in handy.

Finally, I want to thank all those who shared their thoughts and feelings with me while researching and writing this month's column. Special thanks have to go out to Matthew Weier O'Phinney, Matt Brunt, Tom Oram, Simon Holywell, Bjoern M. Wibbenm, Andreas Heigl, Matthew Trask, Marco Pivetta, Gary Hockin, Artūrs Krapāns, Oscar Merida, and Eli White. Couldn't have done it without you all.

Until next month—happy experimenting!

Matthew Setter is an independent software developer, specializing in creating test-driven applications, and technical writer http://www.matthewsetter.com/services/. He's also editor of Master Zend Framework, which is dedicated to helping you become a Zend Framework master? Find out more http://www.masterzendframework.com/welcome-from-phparch.

To Scan or Not to Scan

Chris Cornutt

In the time I've been focusing on security, specifically application security, in the world of PHP there's one question that keeps coming back up over and over again. Developers ask me if there's any way to include an automated step in their deployment process which can help them secure their code. While I'm much more of a fan of code reviews and building security in from the start, there are some things you can do to help catch the more major "gotchas" related to securing your code. In this installment of *Security Corner*, I want to talk about a tool which can help catch some simpler security issues: a security scanner.

When I say "security scanner" here I'm focusing on one particular type: static analysis. In case you're not sure what static scanning is and how it differs from dynamic scanning, here's a quick review:

- Static scanners test the code "at rest" and look for potentially hazardous issues by using pattern matching and flow evaluations. These require direct access to the source code of the application.
- Dynamic scanners are just the opposite—they focus on testing as if they were a user of the system. They're usually pointed at a test instance of the application where they can scan through common issues and endpoints, trying to find security issues by executing with randomized data.

They are two very different kinds of scanning but both have their value. Obviously, dynamic scanning isn't going to find those random code paths not executed by the current code, but static scanning won't find issues which only happen at runtime. For the sake of this article, however, I'm going to focus on static scanners as they're the "closest to the code" and easiest to integrate into a deployment flow.

Scanners in PHP

If you've ever looked around the PHP ecosystem for a security scanner, you've noticed one thing: there's just not a lot of good options out there. As with most open-source software, the quality of the scanners varies widely between packages. Some aren't much more than overly complex `grep` evaluations. Others do a bit of parsing but don't get overly complex. Sadly, when most people ask me about an open-source, free-as-in-beer scanner they can integrate into their development cycle, I don't have a good option to give them.

Several have been created in the past but most of have fallen into disrepair, especially with the release of PHP 7. Sure, there are other scanners with different intentions—like phan[1]—but ones with security PHP intents are pretty hard to track down. A quick Google for "PHP security scanner" turns up a few results including:

- phpscanner[2]
- php-malware-finder[3]
- securityscanner[4]

When you look closely, however, most of these projects haven't had an update in quite a while. It's an unfortunate fact—most developers working on security scanners haven't stuck with it. There are, however, exceptions, thankfully. One example is the RIPS[5] scanner created by *Johannes Dahse*. This scanner, one of the first widely available for PHP, was originally designed to work with PHP 4 (yep, way back) and, unfortunately, doesn't play nicely with most modern PHP 5+ code. While this may sound like bad news, *Johannes* picked his work back up on RIPS and created a commercial service out of it: RIPSTech.com[6]. This service takes the ideas behind RIPS and applies additional logic including taint analysis and method chaining to provide more accurate results. Unfortunately for those looking for an open-source, free-as-in-beer scanner, this isn't the option for you. It's a paid service, but it does have the benefit of being directly targeted at PHP.

There are some other commercial options with a decent level of PHP support. In the security world, PHP is still regarded as "one of those languages" which people serious about security just don't use. As a result there aren't many commercial services supporting PHP evaluation. Some, like Veracode[7], do a good job with their commercial scanner PHP support, but others have yet to add it to their list of supported languages.

Additionally, these off-site options usually have another major downfall—the time it takes for them to scan your code. Also, the larger your codebase the longer the scanner takes to run. This can put a bit of a crimp in the style of most continuous deployment processes, where more instant

1 *phan:* https://github.com/etsy/phan

2 *phpscanner:* https://github.com/Te-k/phpscanner

3 *php-malware-finder:* https://github.com/nbs-system/php-malware-finder/

4 *securityscanner:* https://sourceforge.net/projects/securityscanner/

5 *RIPS:* http://rips-scanner.sourceforge.net

6 *RIPSTech.com:* https://www.ripstech.com

7 *Veracode:* https://www.veracode.com

feedback is needed. Unfortunately, most of the scanning I've seen on larger codebases can take anywhere from a few hours to (yikes!) days. Not exactly the continuous feedback cycle most developers are looking for, eh?

With all of that out of the way, let's look at the types of static scanning and what each has to offer.

Types of Static Scanners

When focusing on static scanning technologies, there are two main categories they fall into—scanners looking for *markers* in your code and scanners performing *flow analysis* to check and see if values are escaped/filtered before use. I personally worked on a scanner which performed the first type of scanning: the psecio/parse[8] package. This made use of the excellent PhpParser[9] library to evaluate the code under review and perform more complex matching than just the simple "grep" could accomplish.

For example, you can use it to parse a simple PHP script like this:

```
<?php
echo 'foobar!';
?>
```

And get back objects representing the structure of the script broken down into an abstract syntax tree:

```
Array
(
   [0] => PhpParser\Node\Stmt\Echo_ Object (
      [exprs] => Array (
         [0] => PhpParser\Node\Scalar\String_ Object (
            [value] => foobar!
            [attributes:protected] => Array (
               [startLine] => 1
               [endLine] => 1
               [kind] => 2
            )
         )
      )
      [attributes:protected] => Array (
         [startLine] => 1
         [endLine] => 1
      )
   )
)
```

This might seem a little daunting if you're not familiar with syntax trees, but read through the output above and you will see how the pieces fit together. The PhpParser library breaks down the script and provides useful, nested information about the contents of the script. This makes it much easier to work with its contents than having to try a series of complicated grep statements.

This kind of system also allows you do to interesting evaluations like: "if an echo uses concatenation and the variable is a superglobal…" This is actually one example of a potential security vulnerability. Using superglobals directly in output is a big no-no as users would then have access to directly output whatever content they want, potentially leading to a cross-site scripting vulnerability.

If you look closely at this situation, though, you'll notice one thing is missing. While this kind of checking lets us look for patterns matching known bad practices, it doesn't really let us know what's happening in the script at the time. What if you've already filtered and validated the value(s) you're using from the superglobal? Would the match in this case be wrong? That's where the next kind of scanning method comes in to help provide more context and correctness in the results.

While the *pattern matching* method of scanning can be useful, the *flow analysis* method is definitely more beneficial. Why is it more beneficial? There's a simple answer—false positives. With the pattern-matching method, the scanner is just looking at the structure of the code to find issues, not what it would behave like during execution. Flow analysis scanners, however, break down the file(s) in your application and create sets of interlinked objects to represent the execution flow of your code. They can then start at one place, like our "echo with superglobal" example from earlier, and work backward to see if there's anything along the way doing validation filtering. In this case, say we've added filtering and validation, despite finding the same pattern the static scan found, the flow scanner can tell there's no actual risk here as output issues have been prevented.

There's a great deal of theory and various concepts supporting flow analysis scanners which I won't get into here, but there's plenty of documentation out there on the web if you're interested in finding out more. I'd suggest looking into Anthony Ferrara's php-cfg[10] tool to do the work of determining the process flow of your current code.

Do You Need a Scanner?

What does this all boil down to? Are static security scanners worth using if they're just going to flag a lot of false positives which you have to spend time tracking down? Well, the answer is "yes." While false positives can be a hassle to track down and potentially mark off the list as invalid, they still add one more layer of protection to your code which wasn't there before.

Developers all make mistakes—we're human, after all. All it takes is one poorly escaped piece of output to introduce a whole host of vulnerabilities. This is exactly the kind of check scanners provide, saving you from issues in the future and giving you a better window into the security-related quality of your code.

For the last 10+ years, Chris has been involved in the PHP community. These days he's the Senior Editor of PHPDeveloper.org and lead author for Websec.io, a site dedicated to teaching developers about security and the Securing PHP ebook series. He's also an organizer of the DallasPHP User Group and the Lone Star PHP Conference and works as an Application Security Engineer for Salesforce. @enygma

8 *psecio/parse:* https://github.com/psecio/parse

9 *PhpParser:* https://github.com/nikic/PHP-Parser

10 *php-cfg:* https://github.com/ircmaxell/php-cfg

October Happenings

PHP Releases

PHP 7.0.12: http://php.net/archive/2016.php#id2016-10-13-1
PHP 5.6.27: http://php.net/archive/2016.php#id2016-10-14-1
PHP 7.1.0RC4: http://php.net/archive/2016.php#id2016-10-19-1

News

Slack Engineering Blog: Taking PHP Seriously

On the Slack Engineering blog there's a new post from one of their engineers talking about a choice the company made about their platform—they decided to take PHP seriously. In this post author Keith Adams talks about why they chose PHP and what kind of experiences they've had with it in their own environment. He starts with some background on the history of PHP itself, where the language came from and what kinds of issues it tries to mainly solve. He then gets into some of what he sees are the "virtues of PHP". http://phpdeveloper.org/news/24500

Medium.com: 10 Modern Software Over-Engineering Mistakes

In this recent post to his Medium.com site Subhas Dandapani shares what he sees as the top ten modern software over-engineering mistakes developers make in modern application development. There are definitely some things in here that are a bit debatable, but development has and will always have a file line between over-engineering and "just the right amount" of work. http://phpdeveloper.org/news/24518

Dailymotion.com: PHP 7 Deployment at Dailymotion

On the Dailymotion.com Engineering blog there's a recent post detailing their experiences moving their services to PHP 7 and the discoveries they made along the way. They started out by looking into Facebook's HHVM project to potentially replace the default PHP interpreter with a better performing core. They mention incompatibilities they discovered and the results in testing it on a handful of servers in production. They had some time to play with things so they waited until PHP 7 was officially released and tried that to make an equal comparison. http://phpdeveloper.org/news/24513

Symfony Finland: What's in Store for PHP Performance?

On the Symfony Finland blog there's a new post looking ahead at what they see in store for PHP's performance in a post-PHP 7.0.x world. These other "unbeaten paths" they mention include a trend towards using asynchronous patterns and the use of application servers (long running PHP processes). http://phpdeveloper.org/news/24510

Matt Stauffer: The New Notification System in Laravel 5.3

In the latest part of his series covering Laravel 5.3, Matt Stauffer has posted this new tutorial covering the new notification system in the latest version of the popular Laravel framework. This notification system makes it simpler to send messages to your user when you don't care as much how they get it, just that they do. He walks you through the creation of your first notification class and breaks it down into its main parts, explaining each one. http://phpdeveloper.org/news/24521

Adam Wathan: Replacing Mocks with Spies

In this post to his site Adam Wathan shares a unit testing tip that can help you with more correct verification in your testing—replacing mocks with spies. He gives a Laravel-based example of using Mockery to set an "expects" call on a method to ensure it's correctly called. He points out, however, that this method is more useful for checking the result of the method call and not really the fact that it was called (a slight but interesting difference). http://phpdeveloper.org/news/24496

Exakat Blog: 6 Good Practices for "Use" in PHP

On the Exakat blog there's a new post sharing six good practices for "use" in PHP. The `use` keyword has a few different places it is used in PHP (like importing namespaced classes and passing in values to closures). Each of the six tips they share come with a bit of explanation and code to back them up. http://phpdeveloper.org/news/24501

Stomt Blog: Shared Components Across Multiple Laravel/Lumen Micro-Services

On the Stomt blog today there's a post showing how you can share components across Laravel/Lumen applications using a simple structure and making things like microservices easier. These microservices required similar functionality and splitting those out into shared components made sense. They walk you through the basic requirements they had when splitting the application and how the components are structured. http://phpdeveloper.org/news/24529

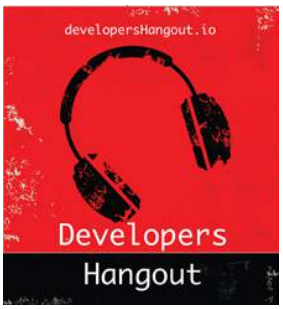
Welcome to php[architect]'s new MarketPlace! MarketPlace ads are an affordable way to reach PHP programmers and influencers. Spread the word about a project you're working on, a position that's opening up at your company, or a product which helps developers get stuff done—let us help you get the word out! Get your ad in front of dedicated developers for as low as $33 USD a month.

To learn more and receive the full advertising prospectus, contact us at ads@phparch.com today!

phproundtable

The PHP podcast where everyone chimes in.

www.phproundtable.com



Listen to developers discuss topics about coding and all that comes with it.
www.developershangout.io

Kara Ferguson
Editorial

Refactoring your words, while you refactor your code.

@KaraFerguson
karaferguson.net

technology : running : programming
rungeekradio.com

The Frederick Web Technology Group
meetup.com/FredWebTech

The PHP user group for the DC Metropolitan area
meetup.com/DC-PHP

Can We Be Nice to Each Other?

Eli White

All right everyone. I know that I've ranted on a similar topic in the past, but it's time to get real. We, as the greater geek & developer community, really need to start being nice to each other.

To be fair, this is not a problem unique to our community. In fact, it's a much bigger problem in society right now. Even if you aren't from the USA, you've probably heard about a particular presidential election that is currently happening here. By the time you are reading this, there's a chance that the election has already happened (but I'll predict that the issues with the election will not have subsided yet).

> *"Don't Be a Dick"*
> *— Wil Wheaton*

If you are on social media, you now probably know what I'm talking about. We've lost the ability in modern society to be nice to each other. We've lost the ability to have pleasant conversations while having a differing opinion. Instead, you are right, they are wrong, and "they" need put in their place. As a society, we need to stop doing this. We need to start following Wheaton's Law[1]:

While we can't fix the whole of society, we can start doing it within our community. One of the biggest things that we can stop doing is immediately discounting someone's choice of tooling, or framework, or programming language.

If someone prefers using a tool that you don't use. Don't dismiss them. Don't tell them it sucks. Respond with "That's interesting, why do you prefer it?" Learn a bit about their point of view, and perhaps provide your view as a counterpoint so that they might be able to learn as well. Do it politely.

Too often I find someone posting something online such as "Was using the Safari Web Inspector today." But what follows is not a conversation, someone trying to help that person solve the problem they were having, or even a gentle: "Oh really, that's interesting, I'm curious why you choose to use Safari for that, I might have some suggestions for you based upon your use case."

Instead, we get statements like: "Well Safari's Web Inspector sucks!" Worse this information may come when someone is excited about finding a new-to-them tool, and then they are immediately blasted for sharing something publicly.

Let's not do that. Let's have a pleasant, welcoming community where we all learn from each other, and share our views while listening to someone's else with equal (or greater) focus.

In short, listen to Wil Wheaton, and don't be a dick.

Eli White is the Conference Chair for php[architect] and Vice President of One for All Events, LLC. He knows that he has broke Wheaton's Law in the past, and is going to try really hard to not do so in the future. @EliW

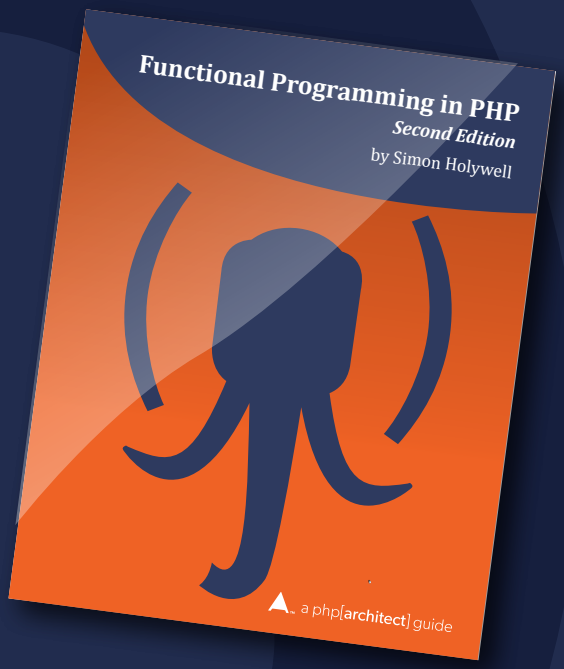
1 *Wheaton's Law:* https://youtu.be/t3ICRoe-QIw

Functional Programming in PHP

Second Edition

by

Simon Holywell



Many languages have embraced Functional Programming paradigms to augment the tools available for programmers to solve problems. It facilitates writing code that is easier to understand, easier to test, and able to take advantage of parallelization making it a good fit for building modern, scalable solutions.

PHP introduced anonymous function and closures in 5.3, providing a more succinct way to tackle common problems. More recent releases have added generators and variadics which can help write more concise, functional code. However, making the mental leap from programming in the more common imperative style requires understanding how and when to best use lambdas, closures, recursion, and more. It also requires learning to think of data in terms of collections that can be mapped, reduced, flattened, and filtered.

Functional Programming in PHP will show you how to leverage these new language features by understanding functional programming principles. With over twice as much content as its predecessor, this second edition expands upon its predecessor with updated code examples and coverage of advances in PHP 7 and Hack. Plenty of examples are provided in each chapter to illustrate each concept as it's introduced and to show how to implement it with PHP. You'll learn how to use map/reduce, currying, composition, and more. You'll see what external libraries are available and new language features are proposed to extend PHP's functional programming capabilities.

Buy Your Copy Today!

http://phpa.me/functional-programming-in-php-2/

Web Security 2016

from php[architect] magazine

Edited by Oscar Merida

Are you keeping up with modern security practices? The ***Web Security 2016 Anthology*** collects articles first published in php[architect] magazine. Each one touches on a security topic to help you harden and secure your PHP and web applications. Your users' information is important, make sure you're treating it with care.

This *Web Security 2016* Book includes:

- An overview of the attacks you should be familiar with and how to protect against exploits.
- Using a PHP-based Intrusion Detection System to monitor and reject requests that attempt to breach your site.
- How to protect against SQL Injection from user-supplied data by using prepared statements.
- A case study in how the Drupal security team keeps core and contributed modules safe.
- How to securely store passwords and understanding the techniques used to crack credentials.
- Using OAuth 2.0 to connect to web services and fetch information for your users without asking for a password.
- How web service security differs from traditional web application security and advice for effectively protecting one from malicious users.
- Identifying the right kind of cryptography to implement in your application and doing it correctly.

Purchase Book

http://phpa.me/web-security-2016

Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

php[**architect**]

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe