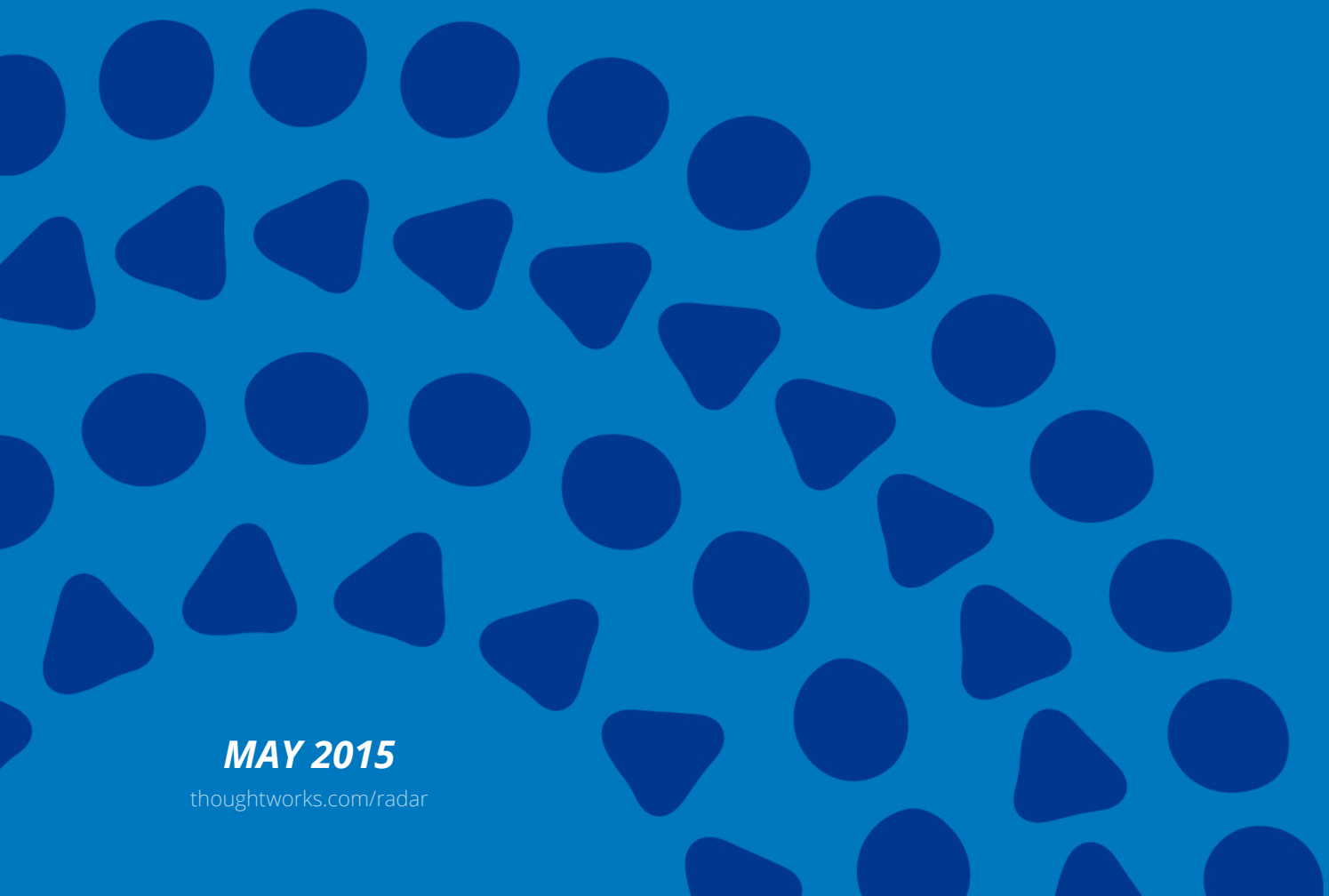
ThoughtWorks®

TECHNOLOGY RADAR

Our thoughts on the technology and trends that are shaping the future



MAY 2015

thoughtworks.com/radar

WHAT'S NEW?

Here are the trends highlighted in this edition:

INNOVATION IN ARCHITECTURE

Organizations have accepted that "cloud" is the de-facto platform of the future, and the benefits and flexibility it brings have ushered in a renaissance in software architecture. The disposable infrastructure of cloud has enabled the first "cloud native" architecture, microservices. Continuous Delivery, a technique that is radically changing how tech-based businesses evolve, amplifies the impact of cloud as an architecture. We expect architectural innovation to continue, with trends such as containerization and software-defined networking providing even more technical options and capability.

A NEW WAVE OF OPENNESS AT MICROSOFT

Whilst Microsoft has dabbled in open-source in the past—including their open-source hosting platform CodePlex—the company's core assets continued to be proprietary and closely guarded secrets. Now, though, Microsoft seems to be embracing a new strategy of openness, releasing large parts of the .NET platform and runtime as open-source projects on GitHub. We're hopeful that this could pave the way to Linux as a hosting platform for .NET, allowing the C# language to compete alongside the current bevy of JVM-based languages.

SECURITY STRUGGLES CONTINUE IN THE ENTERPRISE

Despite increased attention on security and privacy, the industry hasn't made much progress since the last Radar and we continue to highlight the issue. Developers are responding with increased security infrastructure and tooling, building automated test tools such as the Zed Attack Proxy into deployment pipelines. Such tools are of course only part of a holistic approach to security, and we believe all organizations need to "raise their game" in this space.

CONTRIBUTORS

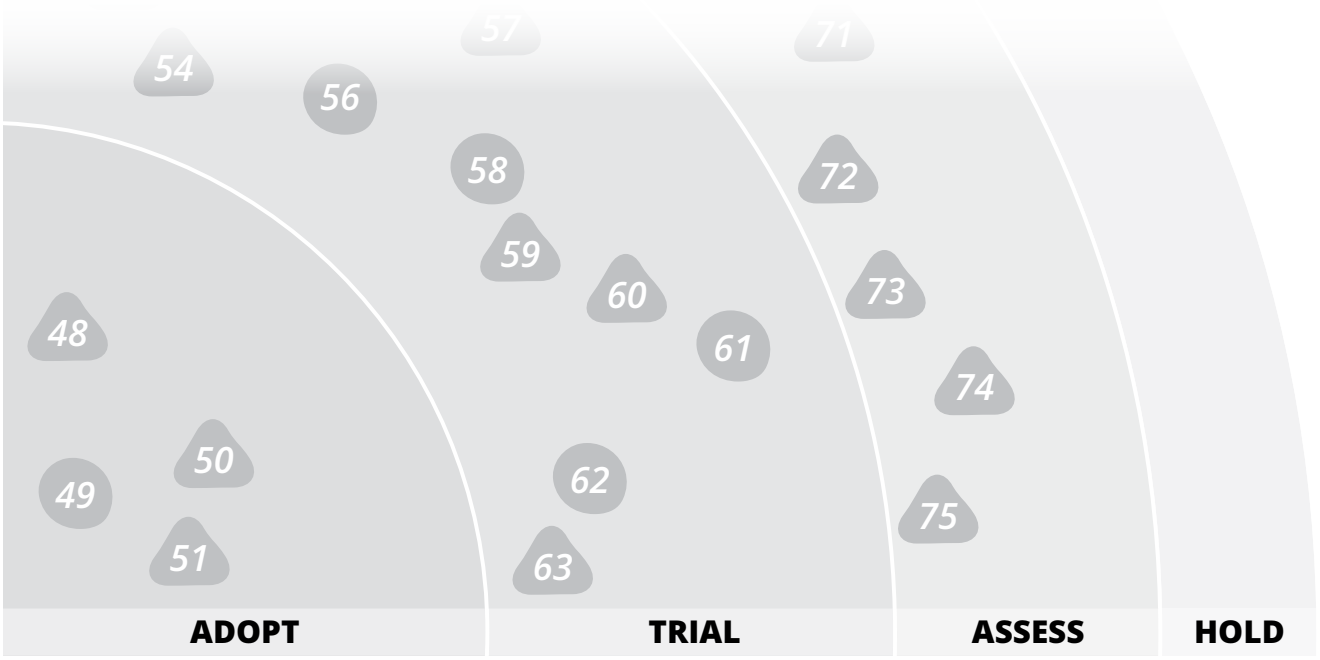
The Technology Radar is prepared by the ThoughtWorks Technology Advisory Board, comprised of:

Rebecca Parsons (CTO)
Martin Fowler(Chief Scientist)
Anne J Simmons
Badri Janakiraman
Brain Leke
Claudia Melo
Dave Elliman
Erik Doernenburg
Evan Bottcher
Hao Xu
Ian Cartwright
James Lewis
Jeff Norris
Jonny LeRoy
Mike Mason
Neal Ford
Rachel Laycock
Sam Newman
Scott Shaw
Srihari Srinivasan
Thiyagu Palanisamy

ABOUT THE TECHNOLOGY RADAR

ThoughtWorkers are passionate about technology. We build it, research it, test it, open source it, write about it, and constantly aim to improve it – for everyone. Our mission is to champion software excellence and revolutionize IT. We create and share the ThoughtWorks Technology Radar in support of that mission. The ThoughtWorks Technology Advisory Board, a group of senior technology leaders in ThoughtWorks, creates the radar. They meet regularly to discuss the global technology strategy for ThoughtWorks and the technology trends that significantly impact our industry.

The radar captures the output of the Technology Advisory Board's discussions in a format that provides value to a wide range of stakeholders, from CIOs to developers. The content is intended as a concise summary. We encourage you to explore these technologies for more detail. The radar is graphical in nature, grouping items into techniques, tools, platforms, and languages & frameworks. When radar items could appear in multiple quadrants, we chose the one that seemed most appropriate. We further group these items in four rings to reflect our current position on them. The rings are:



We feel strongly that the industry should be adopting these items. We use them when appropriate on our projects.

Worth pursuing. It is important to understand how to build up this capability. Enterprises should try this technology on a project that can handle the risk.

Worth exploring with the goal of understanding how it will affect your enterprise.

Proceed with caution.

Items that are new or have had significant changes since the last radar are represented as triangles, while items that have not moved are represented as circles. We are interested in far more items than we can reasonably fit into a document this size, so we fade many items from the last radar to make room for the new items. Fading an item does not mean that we no longer care about it.

For more background on the radar, see thoughtworks.com/radar/faq

THE RADAR

TECHNIQUES

ADOPT
1. Consumer-driven contract testing new
2. Focus on mean time to recovery
3. Generated infrastructure diagrams new
4. Structured logging

TRIAL
5. Canary builds
6. Datensparsamkeit
7. Local storage sync
8. NoPSD
9. Offline first web applications new
10. Products over projects new
11. Threat Modelling new

ASSESS
12. Append-only data store
13. Blockchain beyond bitcoin
14. Enterprise Data Lake
15. Flux new
16. Git based CMS/Git for non-code new
17. Phoenix Environments new
18. Reactive Architectures new

HOLD
19. Long lived branches with Gitflow
20. Microservice envy
21. Programming in your CI/CD tool
22. SAFe™
23. Security sandwich
24. Separate DevOps team

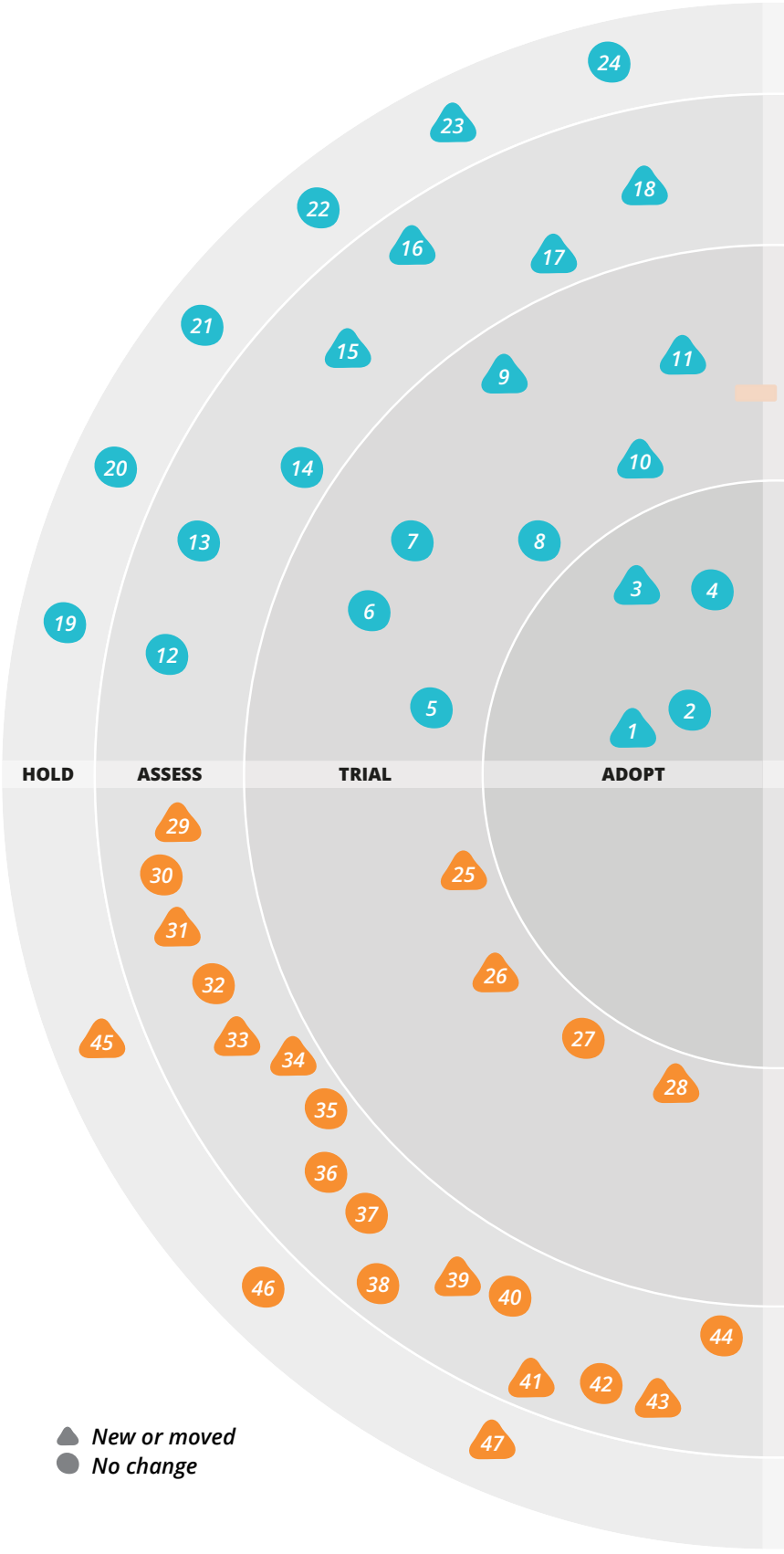
PLATFORMS

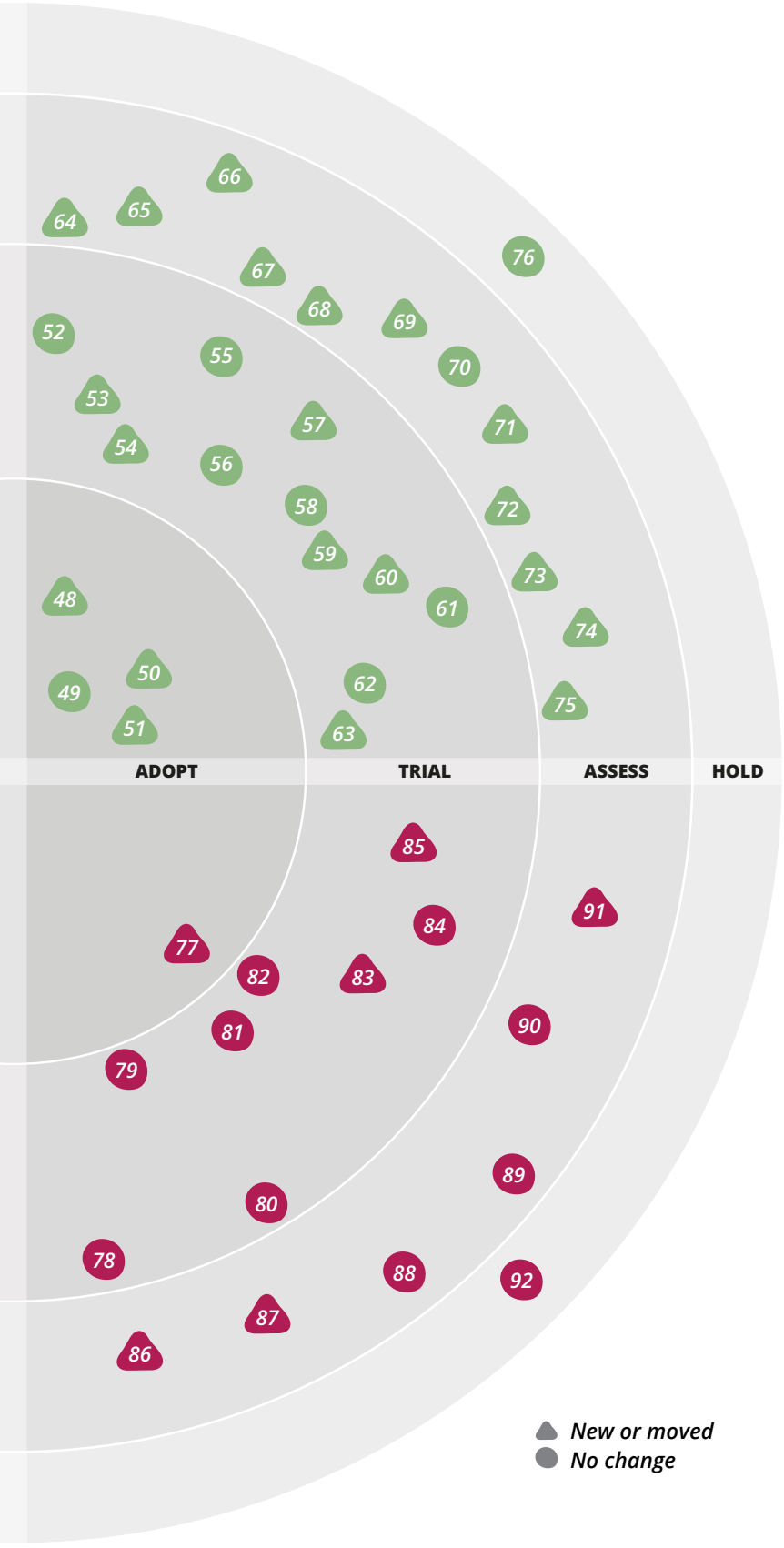
ADOPT

TRIAL
25. Apache Spark
26. Cloudera Impala new
27. DigitalOcean
28. TOTP Two-Factor Authentication

ASSESS
29. Apache Kylin new
30. Apache Mesos
31. CoreCLR and CoreFX new
32. CoreOS
33. Deis new
34. H2O new
35. Jackrabbit Oak
36. Linux security modules
37. MariaDB
38. Netflix OSS Full stack
39. OpenAM
40. SDN
41. Spark Photon/Spark Electron new
42. Text it as a service / Rapidpro.io
43. Time series databases new
44. U2F

HOLD
45. Application Servers new
46. OSGi
47. SPDY new





THE RADAR

TOOLS

ADOPT
48. Composer
49. Go CD
50. Mountebank
51. Postman

TRIAL
52. Boot2docker
53. Brighter new
54. Consul
55. Cursive
56. GitLab
57. Hamms new
58. IndexedDB
59. Polly new
60. REST-assured new
61. Swagger
62. Xamarin
63. ZAP new

ASSESS
64. Apache Kafka new
65. Blackbox
66. Bokeh/Vega new
67. Gor new
68. NaCl new
69. Origami new
70. Packetbeat
71. pdfmake new
72. PlantUML new
73. Prometheus new
74. Quick new
75. Security Monkey new

HOLD
76. Citrix for development

LANGUAGES & FRAMEWORKS

ADOPT
77. Nancy

TRIAL
78. Dashing
79. Django REST
80. Ionic Framework
81. Nashorn
82. Om
83. React.js
84. Retrofit
85. Spring Boot

ASSESS
86. Ember.js new
87. Flight.js
88. Haskell Hadoop library
89. Lotus
90. Reagent
91. Swift

HOLD
92. JSF

TECHNIQUES

When two independently developed services are collaborating, changes to the supplier's API can cause failures for all its consumers. Consuming services usually cannot test against live suppliers since such tests are slow and brittle (martinfowler.com/articles/nonDeterminism.html#RemoteServices), so it's best to use Test Doubles (martinfowler.com/bliki/TestDouble.html), leading to the danger that the test doubles get out of sync with the real supplier service. Consumer teams can protect themselves from these failures by using integration contract tests (martinfowler.com/bliki/IntegrationContractTest.html) – tests that compare actual service responses with test values. While such contract tests are valuable, they are even more useful when consuming services provide these tests to the supplier, who can then run all their consumers' contract tests to determine if their changes are likely to cause problems – adopting consumer-driven contracts (martinfowler.com/articles/consumerDrivenContracts.html). Such **consumer-driven contract tests** are an essential part of a mature microservice testing (martinfowler.com/articles/microservice-testing/) portfolio.

When we need a diagram that describes the current infrastructure or physical architecture we usually take to our favorite technical diagramming tool. If you are using the cloud or virtualization technologies this no longer makes sense, we can use the provided APIs to interrogate the actual infrastructure and generate a live, **automated infrastructure diagram** using simple tools like GraphViz (graphviz.org) or by outputting SVG.

Offline first web applications provide the ability to design web applications for offline access by employing caching and updating mechanisms. The implementation requires a flag in the DOM to check whether the accessing device is offline or online, accessing local storage when offline, and synchronising data when online. All the major browsers now support an offline mode, with the local information accessible by specifying a manifest attribute in the html, which bootstraps the process of downloading and caching the resources such as HTML, CSS, JavaScript, images and other kinds of resources. There are some tools which help simplify offline first implementation such as **Hoodie** (hood.ie), and **CouchDB** (couchdb.apache.org) also offers ability to work with a locally deployed application on a local data storage.

Most software development efforts are done using the mental model of a project, something that is planned, executed, and delivered within defined time-slots. Agile development challenged much of this model, replacing an up-front determination of requirements with an on-going discovery process that runs concurrently with development. Lean startup techniques, such as A/B testing of observed requirements (martinfowler.com/bliki/ObservedRequirement.html), further erode this mindset. We consider that most software efforts should follow the lead of Lean Enterprise (info.thoughtworks.com/lean-enterprise-book.html) and consider themselves to be building products that support underlying business processes. Such products do not have a final delivery, rather an on-going process of exploring how best to support and optimize that business process which continues as long as the business is worthwhile. For these reasons we encourage organizations to think in terms of **products rather than projects**.

ADOPT
1. Consumer-driven contract testing
2. Focus on mean time to recovery
3. Generated infrastructure diagrams
4. Structured logging

TRIAL
5. Canary builds
6. Datensparsamkeit
7. Local storage sync
8. NoPSD
9. Offline first web applications
10. Products over projects
11. Threat Modelling

ASSESS
12. Append-only data store
13. Blockchain beyond bitcoin
14. Enterprise Data Lake
15. Flux
16. Git based CMS/Git for non-code
17. Phoenix Environments
18. Reactive Architectures

HOLD
19. Long lived branches with Gitflow
20. Microservice envy
21. Programming in your CI/CD tool
22. SAFe™
23. Security sandwich
24. Separate DevOps team

TECHNIQUES *continued*

At this point the vast majority of development teams are aware of the importance of writing secure software and dealing with their users' data in a responsible way. They do face a steep learning curve and a vast number of potential threats, ranging from organized crime and government spying to teenagers who attack systems "for the lulz". **Threat Modelling** (owasp.org/index.php/Category:Threat_Modeling) is a set of techniques, mostly from a defensive perspective, that help understand and classify potential threats. When turned into "evil user stories" this can give a team a manageable and effective approach to making their systems more secure.

Flux (facebook.github.io/flux) is an application architecture that Facebook has adopted for its web application development. Usually mentioned in conjunction with **react.js**, Flux is based on a one-way flow of data up through the rendering pipeline triggered by users or other external events modifying data stores. It's been a while since we've seen any alternatives to the venerable model-view-* architectures and Flux embraces the modern web landscape of client-side JavaScript applications talking to multiple back-end services.

These days, most software developers are used to working with Git for source code control and collaboration. But Git can be used as a base mechanism for other circumstances where a group of people need to collaborate on textual documents (that can easily be merged). We've seen increasing amounts of projects use **Git** (git-scm.com) as the basis for a lightweight **CMS**, with text-based editing formats. Git has powerful features for tracking changes and exploring alternatives, with a distributed storage model that is fast in use and tolerant of networking issues. The biggest problem with wider adoption is that Git isn't very easy to learn for non-programmers, but we expect to see more tools that build on top of the core Git plumbing. Such tools simplify the workflow for specific audiences, such as content authors. We would also welcome more tools to support diffing and merging for non-textual documents.

The idea of phoenix servers (martinfowler.com/bliki/PhoenixServer.html) is now well established and has brought many benefits when applied to the right kinds of problems, but what about the environment we deploy these servers into? The concept of **Phoenix Environments** can help. We can use automation to allow us to create whole environments, including network configuration, load balancing and firewall ports, for example by using **CloudFormation** in AWS. We can then prove that the process works, by tearing the environments down and recreating them from scratch on a regular basis. Phoenix Environments can support provisioning new environments for testing, development, UAT and so on. They can also simplify the provision of a disaster recovery environment. As with Phoenix Servers this pattern is not always applicable and we need to think about carefully about things like state and dependencies. Treating the whole environment as a green/blue deployment (martinfowler.com/bliki/BlueGreenDeployment.html) can be one approach when environment reconfiguration needs to be done.

The techniques of functional reactive programming have steadily gained in popularity over recent years, and we're seeing increased interest in extending this concept to distributed systems architectures. Partly inspired by The Reactive Manifesto (reactivemanifesto.org), these **reactive architectures** are based on a one-way, asynchronous flow of immutable events through a network of independent processes (perhaps implemented as microservices). In the right setting, these systems are scalable and resilient and decrease the coupling between individual processing units. However, architectures based entirely on asynchronous message passing introduce complexity and often rely on proprietary frameworks. We recommend assessing the performance and scalability needs of your system before committing to this as a default architectural style.

Traditional approaches to security have relied on up-front specification followed by validation at the end. This **"Security Sandwich"** approach is hard to integrate into Agile teams, since much of the design happens throughout the process, and it does not leverage the automation opportunities provided by continuous delivery. Organizations should look at how they can inject security practices throughout the agile development cycle. This includes: evaluating the right level of Threat Modeling to do up-front; when to classify security concerns as their own stories, acceptance criteria, or cross-cutting non-functional requirements; including automatic static and dynamic security testing into your build pipeline; and how to include deeper testing, such as penetration testing, into releases in a continuous delivery model. In much the same way that DevOps has recast how historically adversarial groups can work together, the same is happening for security and development professionals. (But despite our dislike of the Security Sandwich model, it is much better than not considering security at all, which is sadly still a common circumstance.)

PLATFORMS

Apache Spark (spark.apache.org) has been steadily gaining ground as a fast and general engine for large-scale data processing. The engine is written in Scala and is well suited for applications that reuse a working set of data across multiple parallel operations. It's designed to work as a standalone cluster or as part of Hadoop YARN cluster. It can access data from sources such as HDFS, Cassandra, S3 etc. Spark also offers many higher level operators in order to ease the development of data parallel applications. As a generic data processing platform it has enabled development of many higher level tools such as interactive SQL (Spark SQL), real time streaming (Spark Streaming), machine learning library (MLib), R-on-Spark etc.

For a while now the Hadoop community has been trying to bring low-latency, interactive SQL capability to the Hadoop platform (better known as SQL-on-Hadoop). This has led to a few open source systems such as Cloudera Impala, Apache Drill, Facebook's Presto etc being developed actively through 2014. We think the SQL-on-Hadoop trend signals an important shift as it changes Hadoop's proposition from being a batch oriented technology that was complementary to databases into something that could compete with them.

Cloudera Impala (cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html) was one of the first SQL-on-Hadoop platforms. It is a distributed, massively-parallel, C++ based query engine. The core component of this platform is the Impala daemon that coordinates the execution of the SQL query across one or more nodes of the Impala cluster. Impala is designed to read data from files stored on HDFS in all popular file formats. It leverages Hive's metadata catalog, in order to share databases and tables between the two database platforms. Impala comes with a shell as well as JDBC and ODBC drivers for applications to use.

Passwords continue to be a poor mechanism for authenticating users and we've recently seen companies such as Yahoo! move to a "no passwords" solution—a one-time code is texted to your phone whenever you need to log in from a new browser. If you are still using passwords we recommend employing **two-factor authentication** which can significantly improve security. Time-based One-Time Password (**TOTP**) (en.wikipedia.org/wiki/Time-based_One-time_Password_Algorithm) is the standard algorithm in this space, with free smartphone authenticator apps from Google (play.google.com/store/apps/details?id=com.google.android.apps.authenticator2) and Microsoft (windowsphone.com/en-us/store/app/authenticator/e7994dbc-2336-4950-91ba-ca22d653759b).

ADOPT

TRIAL
25. Apache Spark
26. Cloudera Impala
27. DigitalOcean
28. TOTP Two-Factor Authentication

ASSESS
29. Apache Kylin
30. Apache Mesos
31. CoreCLR and CoreFX
32. CoreOS
33. Deis
34. H2O
35. Jackrabbit Oak
36. Linux security modules
37. MariaDB
38. Netflix OSS Full stack
39. OpenAM
40. SDN
41. Spark Photon/Spark Electron
42. Text it as a service / Rapidpro.io
43. Time series databases
44. U2F

HOLD
45. Application Servers
46. OSGi
47. SPDY

PLATFORMS *continued*

Apache Kylin (kylin.io) is an open source analytics solution from eBay Inc. that enables SQL based multidimensional analysis (OLAP) on very large datasets. Kylin is intended to be a Hadoop based hybrid OLAP (HOLAP) solution that will eventually support both MOLAP and ROLAP style multidimensional analysis. With Kylin you can define cubes using a Cube Designer and initiate an offline process that builds these cubes. The offline process performs a pre-join step to join facts and dimension tables into a flattened out structure. This is followed by a pre-aggregation phase where individual cuboids are built using Map Reduce jobs. The results are stored in HDFS sequence files and are later loaded into HBase. The data requests can originate from SQL submitted using a SQL-based tool. The query engine (based on **Apache Calcite**), determines if the target dataset exists in HBase. If so, the engine directly accesses the target data from HBase and returns the result with sub-second latency. If not, the engine routes the queries to **Hive** (or any other SQL on Hadoop solution enabled on the cluster).

CoreCLR (github.com/dotnet/coreclr) and **CoreFX** (github.com/dotnet/corefx) is the core platform and framework for .NET. Although not new, they have recently been open sourced by Microsoft. A key change is that these dependencies are bin-deployable, they do not need to be installed on a machine in advance. This eases side-by-side deployments, allowing applications to use different framework versions without conflicts. Something written in .NET is then an implementation detail, you can install a .NET dependency into any environment. A .NET tool is no different than something written in C from an external dependency perspective, making it a much more attractive option for general purpose applications and utilities. CoreFX is also being factored into individual NuGet dependencies, so that applications can pull what they need, keeping the footprint for .NET applications and libraries small and making it easier to replace part of the framework.

Heroku, with its 12-factor application model, has changed the way we think about building, deploying, and hosting web applications. **Deis** (deis.io) encapsulates the Heroku PaaS model in an open-source framework that deploys onto Docker containers hosted anywhere. Deis is still evolving, but for applications that fit the 12-factor model it has the potential to greatly simplify deployment and hosting in the environment of your choice. Deis is yet another example of the rich ecosystem of platforms and tools emerging around Docker.

Predictive analytics are used in more and more products, often directly in end-user facing functionality. **H2O** (docs.0xdata.com) is an interesting new open source package (with a startup behind it) that makes predictive analytics accessible to project teams due to its easy-to-use user interface. At the same time it integrates with the data scientists' favorite tools, R and Python, as well as Hadoop and Spark. It offers great performance and, in our experience, easy integration at runtime, especially on JVM-based platforms.

When Oracle ceased development on Sun's OpenSSO—an open source access management platform—It was picked up by ForgeRock and integrated into their Open Identity Suite. Now named **OpenAM** (forgerock.com/products/open-identity-stack/openam), it fills the niche for a scalable, open-source platform that supports OpenID Connect and SAML 2.0. However, OpenAM's long history has resulted in a sprawling codebase whose documentation can be inscrutable. Hopefully, a slimmed-down alternative with better support for automated deployment and provisioning will emerge soon.

Spark (spark.io) is a full stack solution for cloud connected devices. **Spark Photon** is a microcontroller with wifi module. **Spark Electron** is a variant that connects to a cellular network. Spark OS adds REST API to the devices. This simplifies the entry to IoT and building your own connected devices.

A **time series database** (TSDB) is a system that is optimized for handling time series data. It allows users to perform CRUD operations on various time series organized as database objects. It also provides the ability to perform statistical calculations on the series as a whole. Although TSDBs are not entirely a new technology we are seeing a renewed interest in the these databases primarily in the realm of IoT applications. This is being facilitated by many open source and commercial platforms (such as **OpenTSDB**, **InfluxDB**, **Druid**, **BlueFloodDB** etc.) that have mushroomed recently. Its also worth mentioning that some of these systems use other distributed databases such **Cassandra** and **HBase** as their underlying storage engine.

The rise of containers, phoenix servers and continuous delivery has seen a move away from the usual approach to deploying web applications. Traditionally we have built an artifact and then installed that artifact into an application server. The result was long feedback loops for changes, increased build times and the not insignificant overhead of managing these application servers in production. Many of them are a pain to automate too. Most teams we work with favor bundling an embedded http server within your web application. There are plenty of options available: Jetty, SimpleWeb, Webbit and Owin Self-Host amongst others. Easier automation, easier deployment and a reduction in the amount of infrastructure you have to manage lead us to recommend embedded servers over **application servers** for future projects.

The **SPDY** (chromium.org/spdy/spdy-whitepaper) protocol was developed by Google from 2009 as an experiment to provide an alternative protocol to address performance shortcomings of HTTP/1.1. The new HTTP/2 standard protocol includes many of the key performance features of SPDY, and Google has announced it will drop browser SPDY support in early 2016. If your application requires the features of SPDY, we recommend you look instead at HTTP/2.

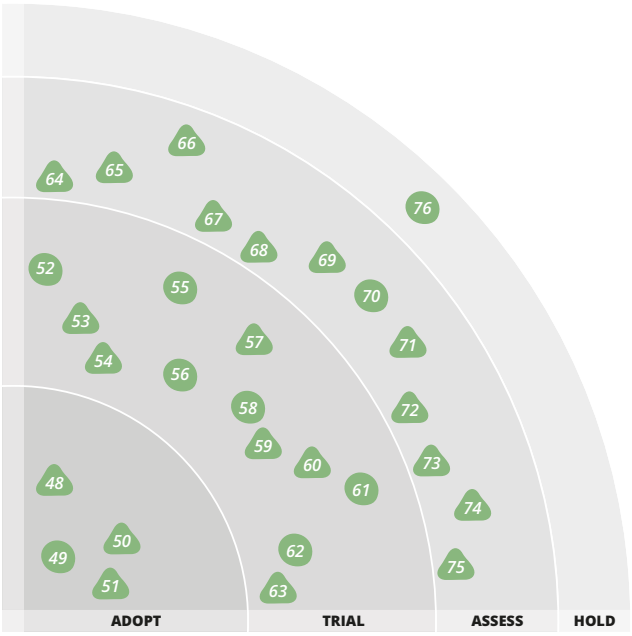
TOOLS

Although the idea of dependency management is not new and considered to be a fundamental development practice, it is not widely adopted by the PHP community. **Composer** (getcomposer.org) is a tool for dependency management in PHP. It is strongly influenced by tools from other technology stacks like Node's npm and Ruby's Bundler. We are now seeing wide adoption across PHP projects and it is fairly mature. You can still have to do some shims for internal libraries, you can use it for most external libraries.

Good testing of components in an enterprise system is critical and with increased emphasis on service-based separation and deployment automation—critical factors for success with microservices—better tooling in this space is needed. The industry term "service virtualization" refers to tools that can emulate specific components in such an environment. We have seen great success with **Mountebank** (mbtest.org), a lightweight tool for stubbing and mocking HTTP, HTTPS, SMTP and TCP.

Postman (getpostman.com/features) is a Chrome extension that acts as a REST client in your browser, allowing you to create requests and inspect responses. It is a useful tool when developing an API or implementing a client to call an existing API. Postman supports OAuth1 and OAuth2 tokens allowing addition of them to requests where necessary. The response is available as a prettified JSON or XML. With Postman you are able to retrieve a history of requests performed to quickly edit and test the API response to different data. It offers a suite of extensions that allow you to use it as a full-blown test runner too, although we discourage the record and replay style of testing it promotes.

Brighter (iancooper.github.io/Paramore/Brighter.html) is an open source library for .Net that provides scaffolding to implement Command Invocation. We have had good feedback from teams using it, especially in conjunction with the ports and adaptors pattern and **CQRS**. They especially like that it integrates well with **Polly** to provide circuit breaking functionality.



We continue to be impressed with **Consul** (consul.io), a service discovery tool supporting both DNS and HTTP-based discovery mechanisms. It goes beyond other discovery tools by providing customizable health-checks for registered services, ensuring that unhealthy instances are marked accordingly. More tools have emerged to work with Consul to make it even more powerful. Consul Template (github.com/hashicorp/consul-template) enables configuration files to be populated with information from Consul, making things like client-side load balancing using mod_proxy much easier. In the world of Docker, registrator (github.com/gliderlabs/registrator) can automatically register docker containers as they appear with Consul with extremely little effort, making it much easier to manage container-based setups.

ADOPT
48. Composer
49. Go CD
50. Mountebank
51. Postman

TRIAL
52. Boot2docker
53. Brighter
54. Consul
55. Cursive
56. GitLab
57. Hamms
58. IndexedDB
59. Polly
60. REST-assured
61. Swagger
62. Xamarin
63. ZAP

ASSESS
64. Apache Kafka
65. Blackbox
66. Bokeh/Vega
67. Gor
68. NaCl
69. Origami
70. Packetbeat
71. pdfmake
72. PlantUML
73. Prometheus
74. Quick
75. Security Monkey

HOLD
76. Citrix for development

TOOLS *continued*

Many many wonderful stories of failure in our industry are caused by the assumption that networks are always reliable and servers respond quickly and correctly all the time. **Hamms** (github.com/kevinburke/hamms) is an interesting open-source tool which acts as a badly behaved HTTP server, triggering a number of failures including connection failures or slow and/or malformed responses. It may be useful for testing that your software handles failures gracefully.

Several of our teams working on .Net projects have recommended **Polly** (github.com/michael-wolfenden/Polly) as being useful when building microservice based systems. It encourages the fluent expression of transient exception handling policies and the circuit breaker pattern including policies such as Retry, Retry Forever and Wait and Retry. Libraries already exist in other languages, Hystrix for Java for example, and Polly is a welcome addition from the .Net community.

REST-assured (code.google.com/p/rest-assured) is a Java domain specific language for testing and validating RESTful services. It simplifies the testing of REST based services built on top of HTTP Builder. REST-assured supports the different REST requests and can be used to validate and verify the responses from the APIs. It also provides a JSON schema validation and can thus be used to verify that the endpoints are returning the right types of expected data.

The **ZED Attack Proxy (ZAP)** (owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project) is a project from OWASP which allows you to probe an existing site for security vulnerabilities in an automated fashion. It can be used as part of periodic security testing, or else integrated into a CD pipeline to provide ongoing checks for common vulnerabilities. The use of a tool like ZAP doesn't replace the need to think carefully about security and do other sorts of more thorough testing, but as another tool to help ensure our systems are more secure it's a good addition to the toolbox.

Many recent developments in enterprise software revolve around asynchronous sequences of immutable event sequences as opposed to synchronous, point-to-point requests that modify state. **Apache Kafka** (kafka.apache.org) is an open-source messaging framework that supports this architectural style by publishing ordered message feeds to many independent, lightweight consumers. Kafka's unique design allows the number of consumers to scale while maintaining strong ordering on the messages.

Blackbox (github.com/StackExchange/blackbox) is a simple tool for encrypting specific files while at rest in your source repository. This is particularly useful if you need to store passwords or private keys. Blackbox works with Git, Mercurial and Subversion and uses GPG for the encryption. Each user has their own key, which makes it easy to revoke access on a granular level. There is a lot happening in this space and a few other players to consider including **git-crypt** and **Trousseau**.

In the world of data science and analytics, much of the work is done using Python and R, languages which sadly offer few options for web-accessible plotting of visualizations. One approach is to convert the result of analysis into something that can be easily visualized and interacted with in the browser. We're aware of two tools that are an attempt to do this. **Bokeh** (bokeh.pydata.org) is a Python and JavaScript library that allows you to create interactive visualizations "in the style of D3.js" but with high performance over large or streaming data sets. **Vega** (trifacta.github.io/vega) is a declarative visualization grammar for D3 that consumes server-generated JSON datasets and translates visualization descriptions into D3.js code.

Gor (github.com/buger/gor) is an open-source tool for capturing and replaying live HTTP traffic into a test environment in order to continuously test your system with real data. It can be used to increase confidence in code deployments, configuration changes and infrastructure changes.

The **NaCl** (nacl.cr.yp.to) library (pronounced 'Salt') provides a set of features for encryption, decryption, and signatures designed to make it easier to implement secure network communication or other cryptography requirements. Although these functions exist in other libraries, NaCl promises higher speed and easier to use APIs. Current support is for C and C++ with Python wrappers in progress.

Origami (facebook.github.io/origami) is a free tool for designing user prototypes with a variety of keyboard shortcuts for common functions. It provides the possibility of exporting the prototypes as code snippets to Objective-C for iOS, Java for Android and JavaScript for Web. This tool can be used to rapidly build interactive user facing prototypes and testing user flows. We recommend investigating this tool if the use case fits from the experience we have gathered from several of our teams.

pdfmake (github.com/bpampuch/pdfmake) is a JavaScript library which allows for creation and printing of PDF documents directly in the browser. To use pdfmake you construct a document object that supports structural elements such as tables, columns, and rich styling, then helper methods can create and print or download a PDF without leaving client-side JavaScript.

Developing a software system by first creating a large number of detailed diagrams is an approach that, in our experience, does not compare favorably to the alternatives. However, describing a particularly complex and intricate part of the system with a diagram is usually a good idea, and the UML itself offers a number of useful and commonly understood diagrams. We like **PlantUML** (plantuml.sourceforge.net) for creating these diagrams because it allows expressing the intent behind the diagrams in a clear textual form, without having to fiddle with overloaded graphical tools. Having a textual form also allows versioning and storage alongside the source code.

SoundCloud have recently open sourced a Graphite replacement, **Prometheus** (prometheus.io).Developed as a reaction to difficulties with **Graphite** in their production systems, Prometheus works differently to Graphite, by primarily supporting a pull-based HTTP model (although a more Graphite-like push model is also supported). It also goes beyond Graphite by being built to support alerting based on captured metrics, so it becomes a much more active part of your operational toolset. Some caution should be used in adopting new technology in the production monitoring space, but early reports are that SoundCloud are happy using it in production, and Docker are also contributing to ongoing development.

Quick (github.com/Quick/Quick) is a testing framework for Swift and Objective-C, which comes bundled with **Nimble**, a matcher framework for tests. Quick helps verify the behavior of Swift and Objective-C programs. Quick has the same syntactic flavor as **RSpec** and **Jasmine** and is easy to set up. It is very organized, allows for assertion of types and makes it easy to test asynchronous code.

Security Monkey (github.com/Netflix/security_monkey) is another tool in Netflix's Simian Army, which is a suite of tools designed to ensure that systems are being built in a resilient fashion. As well as providing a (configurable) assessment of any potential security vulnerabilities in your AWS setup, it can also be used to monitor changes on an ongoing basis, alerting different groups as required. It does overlap in some ways with AWS' own Trusted Advisor Report (aws.amazon.com/premiumsupport/trustedadvisor) and CloudTrail (aws.amazon.com/cloudtrail) service, as it was developed prior to both these services being made generally available, but its capabilities do go beyond these offerings. If either of those services don't quite meet your requirements, Security Monkey is worth a look.

LANGUAGES & FRAMEWORKS

Since we last talked about **Nancy** (nancyfx.org) on the technology radar it has become the default choice on our projects. Architectures centred around small, vertical slices and microservices simply require light-weight deployment options and low ceremony tooling.

One benefit to the ongoing avalanche of front-end JavaScript frameworks is that occasionally, a new idea crops up that makes us think. **React.js** (facebook.github.io/react) is a UI/View framework in which JavaScript functions generate HTML in a reactive data flow. We have seen several smaller projects achieve success with React.js and developers are drawn to its clean, composeable approach to componentization.

Spring Boot (projects.spring.io/spring-boot) allows easy set up of standalone Spring-based applications. It's ideal for pulling up new microservices and easy to deploy. It also makes data access less of a pain due to the hibernate mappings with much less boilerplate code. We like that Spring Boot simplifies Java services built with Spring, but have learned to be cautious of the many dependencies. Spring still lurks just beneath the surface.

Widespread usage of AngularJS continues on ThoughtWorks projects, although not every experience is positive. We continue to advise teams to assess whether the additional complexity of a single-page JavaScript application is necessary to meet their requirements. We also recommend assessing alternative frameworks, and in this radar edition we highlight **Ember.js** (emberjs.com) which is growing in popularity within ThoughtWorks. Ember is praised for its approach of opinionated convention over configuration, responsive core team of committers, performance, and build tooling support via Ember CLI.

In the crowded space of JavaScript frameworks, we want to highlight **Flight.js** (flightjs.github.io) as a lightweight framework to build components. Flight gets by without much magic when adding behavior to DOM nodes. Its

event-driven and component-based nature promotes writing decoupled code. This makes testing individual components comparatively easy. Care must be taken, however, when components need to interact with each other. There is little support for testing and a real danger to get into event hell. We do like that it uses functional mixins for behavior, like composition instead of inheritance.

With some real-world experience under our belt, **Swift** (developer.apple.com/swift) still shows a lot of promise. Some of the problems, like long compile times, are being addressed. However, continued language changes cause extra development effort and make building older versions of your own software burdensome. Testing and refactoring also remain painful. On balance, though, you should still consider Swift when starting new development projects for the Apple ecosystem.

ADOPT
77. Nancy

TRIAL
78. Dashing
79. Django REST
80. Ionic Framework
81. Nashorn
82. Om
83. React.js
84. Retrofit
85. Spring Boot

ASSESS
86. Ember.js
87. Flight.js
88. Haskell Hadoop library
89. Lotus
90. Reagent
91. Swift

HOLD
92. JSF

ThoughtWorks is a software company and community of passionate, purpose-led individuals that specialize in software consulting, delivery and products. We think disruptively to deliver technology to address our clients' toughest challenges, all while seeking to revolutionize the IT industry and create positive social change. We make pioneering tools for software teams who aspire to be great. Our products help organizations continuously improve and deliver quality software for their most critical needs. Founded over 20 years ago, ThoughtWorks has grown from a small group in Chicago to a company of over 3000 people spread across 30 offices in 12 countries: Australia, Brazil, Canada, China, Ecuador, Germany, India, Singapore, South Africa, Uganda, the United Kingdom, and the United States.

ThoughtWorks®