

## TECHNOLOGY ASSESSMENT

### Java: Two and a Half Years After the Acquisition

Al Hilwa

#### IDC OPINION

Two and a half years after Oracle closed on the Sun acquisition in January 2010, many of the fears of the broader Java community have not materialized. Oracle has navigated most decisions with a deliberate and decisive approach that should inspire the community's confidence in Java's long-term prospects. In addition:

- ☒ Oracle moved decisively to settle a long-running dispute over the approval of Apache Harmony's implementation of Java by not removing restrictions on the use of the certified product that Apache insisted be removed.
- ☒ Oracle brought key vendors IBM, Apple, and SAP to the OpenJDK open source implementation of Java and anointed OpenJDK as the reference implementation for the technology.
- ☒ Oracle successfully shipped Java Standard Edition (SE) 7 after splitting the improvements planned prior to the acquisition into two releases. As a result, Java made more significant advancements after the Sun acquisition than in the two and half years prior to the acquisition.
- ☒ Oracle advanced more realistic road maps for Java SE 8 and Java Enterprise Edition (EE) 7 through the Java Community Process (JCP). Delivering the two new releases on schedule will be an important validation of Oracle's Java strategy.
- ☒ The Java ecosystem is healthy and remains on a growing trajectory, with more programming languages than ever now hosted on the Java Virtual Machine (JVM) and with many new developers further bolstering the broader Java skills ecosystem as mobile Android developers.
- ☒ Java is under pressure from competing developer ecosystems, including the aggressively managed Microsoft platform ecosystem and the broader Web ecosystem with its diverse technologies and lightweight scripting languages and frameworks. To remain relevant and attractive to new developers, Java must evolve on a faster schedule and effectively support the ongoing industry transformation into mobile, cloud, and social applications.
- ☒ Java is at risk of fragmentation into multiple forks of loosely similar but competing technologies as a result of Google's success with Android and the potential of Android's evolution into client and server form factors.



## **IN THIS STUDY**

On January 27, 2010, Oracle announced the closing of its acquisition of Sun Microsystems after a two-year review process by various regulatory authorities commencing with the April 2009 announcement of the acquisition. The acquisition brought the Java technology under the custodianship of Oracle, leading to community anxiety and consternation around how Oracle might run Java. We take a detailed look in this study at how Java has fared under Oracle.

## **SITUATION OVERVIEW**

In late 2008, IBM approached Sun Microsystems (Sun) about a possible acquisition, kicking off several months of internal discussion, the news of which broke on March 18, 2009. Sun solicited bids from other companies during these discussions, and on April 20, 2009, Oracle and Sun jointly announced that they came to an agreement over the acquisition of Sun by Oracle. A long regulatory review process commenced in the United States and Europe over competitive issues raised by the acquisition, during which the industry and many in the open source community speculated on the effect of this acquisition on the future of Java. Many members of the Java community were concerned that the more software-focused Oracle, which was known for its aggressive tactics with its partners and customers, might run Java in a less open or democratic way, raise licensing costs, or turn away from Sun's prior effort to release Java in open source. In January 2010, Oracle announced the closing of its acquisition of Sun Microsystems after the last of the regulatory hurdles over MySQL were cleared with the European Union. We have since had almost two and a half years to assess the impact of Oracle's tenure as a Java leader.

---

## **The Value of Java**

Oracle had been a key Java supporter from the early days and, like IBM and others, had built a sizable middleware business franchise around Java with a portfolio of middleware products. Oracle had embarked on a spree of major software acquisitions in the five years preceding the Java acquisition, culminating in the acquisition of BEA in April 2008. By the time Oracle made its bid for Java, it was already a top 2 Java middleware player along with IBM. Just as importantly, Oracle had invested in a massive re-architecting — based on Oracle Application Development Framework (ADF) — of its applications product line, itself a melding of multiple acquisitions (e.g., PeopleSoft and Siebel), into an expansive new product stack centered around Java. With this context, the Sun acquisition should be seen as driven by Oracle's desire to own the technology on which Oracle was betting its future.

Oracle has also successfully leveraged Sun's software and hardware to build an integrated appliance strategy. With the Sun acquisition, Oracle was able to add a Java middleware appliance, known as Exalogic, to its already released Exadata server. Oracle's Exa servers have been well received and have enjoyed substantial growth, though much of their generated revenue growth came at the expense of the declining traditional, general-purpose Unix servers. Overall, Oracle appears to have successfully digested Sun, avoided significant hits on its profitability and margins by lifting the value of the hardware business through software integration, and eking out significant software growth through cross-selling to Sun customers. In many ways, acquiring Java has been a strategic masterstroke for Oracle, giving it a stronger command over the market it plays in, and its strategic destiny. The bigger test for Oracle is the overall health of the Java ecosystem, which is where an even bigger chunk of Java value sits.

---

## **Java Community Anxiety**

Over the years, Oracle had developed an aggressive public persona built on its hypercompetitive marketing and sales tactics and often stimulated by the pronouncements of its iconic leader Larry Ellison. Oracle has historically been and is a recognized innovator and leader in database technology, constantly releasing new features and capabilities in its flagship product, but always monetizing them aggressively through high-priced database options (e.g., Oracle's Real Application Clusters or RAC) and license maintenance fees. Oracle was thought by many, especially in the open source software community, to be the antithesis of the type of company that should run Java. As a result, the Sun acquisition raised alarms in the Java community. The concerns were around whether Oracle would keep Java implementations in open source, whether it would seek to monetize Java more aggressively by increasing its licensing fees, or how Oracle would govern Java and whether it would do so with sufficient community input. It is thus important to assess how Oracle has done in this area over the past two and a half years.

Much of the anxiety of the open source community likely stems from the ineffective ways that Sun was able to monetize Java prior to the acquisition. Sun was a company focused on hardware and saw software "as a feature of the hardware," as its CEO was often fond of saying. Sun's focus on hardware was likely an early enabler for Java adoption by many software companies that saw Sun as lacking the will or competency to build a traditional software business around Java. In reality, Sun's hardware focus and lack of experience in building and operating a sizable software business were responsible for a significant amount of dithering and indecisiveness in managing Java and leading its ecosystem. Sun was often torn between disruptive open source instincts and the investments needed to effectively monetize Java through software licensing and never executed decisively or effectively on either. Additionally, Sun's hardware business was in slow decline as the market was shifting toward x86 servers instead of the big iron servers Sun was making and selling. Under Sun's control, Java suffered from slow evolution and a low level of investment.

---

## **Why the JCP Was Created**

The problem of ensuring that a widely used technology platform is not evolved in an autocratic manner by a single vendor is not new for Java. Java was invented and launched by Sun Microsystems as a proprietary product, but its wide and broad adoption made its community nervous about overdependence on a single commercial entity. Thus a governance process known as the Java Community Process was implemented by Sun to ensure that Java is evolved with input and direction from the community. The JCP is in fact a set of formal processes for evolving Java that is unique in the industry in terms of its specific arrangement, scope, and complexity. The process allows Java to be evolved in a way that reconciles the needs of many vendors, most of which are fierce competitors. Indeed, the JCP as a governance process has been instrumental in forging the expansive Java ecosystem and is central to Java's position in the industry, allowing competitors such as Oracle, IBM, SAP, and Red Hat to collaborate on a common interest under one organization. The JCP is not however without its challenges.

---

## **Oracle and Open Source**

It is often underestimated the degree to which Oracle promoted and participated in open source projects prior to the Sun acquisition. Oracle was an active member of the Eclipse Foundation and has supported the IDE aggressively, even as it continued to support its JDeveloper IDE. Another example is Oracle's contribution of TopLink in 2007 to the Eclipse Foundation, which later became the reference implementation for the Java Persistence API 2.0 component. There are many other examples, not least of which are contributions and participation in Linux, PHP, Apache, Eclipse, Berkeley DB, and InnoDB. After the acquisition, Oracle made decisive moves to move Java more aggressively to a full open source code. In mid-2011, Oracle announced that it would base its reference implementation on the open source code in the OpenJDK project. This was a simplification of Oracle's development process and also a step toward a higher level of transparency to the community. With this move, Oracle transformed the OpenJDK project to a more credible initiative that was able to later win collaborators such as IBM and Apple.

In the two and a half years since the acquisition, we are able to say that none of the fears of the open source community materialized. To be sure, Oracle did whittle down Sun's expansive open source portfolio. For example, OpenSolaris was terminated early, and conflicts with the open source communities around projects such as OpenOffice and Hudson ended up in forking the code and eventually handing the code to other open source organizations to manage. Overall, however, Oracle has governed Java in a more decisive and focused fashion. It has continued to monetize Java much like it had been doing prior to the acquisition, through its middleware portfolio (much like Sun had done), but has not significantly increased its licensing fees, which constitute a small and decreasing fraction of Oracle's software revenue. Oracle in fact doubled down on open source by bringing additional partners to the OpenJDK project and deciding to use it as the reference implementations for Java SE. Oracle may have taken some unpopular decisions, such as deciding not to issue a validation kit for the Harmony independent implementation of Java, but its actions were aimed at pushing the JCP to move faster and along a more productive path.

---

## **OpenJDK and the End of Harmony**

As a specification lead, Oracle must deliver reference implementations of Java for new releases. OpenJDK was launched by Sun in 2006 to put the Java SE code in open source under the GNU General Public License (GPL), version 2. While OpenJDK was initially well received, there were Java stakeholders that had started supporting an independent clean-room implementation of Java called Harmony. In particular, IBM and Apache were interested in making Java even more broadly available without licensing fees and without the dominant control of Sun and launched the Harmony project in 2005 to create a compatible clone of Java for release under the more permissive Apache Software License (ASL).

The OpenJDK was historically a project that only Sun maintained, with some help from Red Hat. In late 2010, IBM and Oracle announced that they were collaborating in the OpenJDK community. The move brought together the two largest commercial players in the Java ecosystem around OpenJDK, turning IBM away from supporting the Apache Harmony project. IBM had previously been the primary commercial supporter and funder of Harmony, so the defunding of Harmony led to its retirement to the Apache Attic by the Apache Foundation in late 2011. In mid-2011, Oracle announced that it will be basing its reference implementation on the open source code in the OpenJDK project. This was a simplification of Oracle's development process and also a step toward a higher level of transparency to the community. Previously, the commercial Sun JDK was the reference implementation that gave Sun an advantage over other implementations — Oracle leveled the playing field by making the OpenJDK implementation the reference implementation to which its own commercial implementation has to conform in the same manner as other vendors. With this move, Oracle transformed the OpenJDK project to a more credible initiative that is able to win collaborators such as IBM and Apple.

The resolution of formal efforts to create unlicensed clones of Java was a decisive move by Oracle to keep the community aligned along a single compatible version of Java. IDC believes this was a positive step in the evolution of Java as it helped focus community energy on the evolution of a single version under a single governance body and brand, avoid fragmentation, and preserve licensing fees, which are the most direct monetization revenue streams to ensure investment by Oracle to Java. The move does not guarantee that similar technologies that are partially or totally compatible with Java will not evolve and does not protect against the fragmentation of the Java ecosystems. It does, however, ensure that anything called Java is evolved and managed under the largely democratic JCP.

---

## **Competition and the Speed of JCP**

The key problem with the JCP has historically been the level of overhead it imposes on the decision process and the tax in timeliness that such overhead extracts. Java does not operate in a vacuum. There are many ongoing efforts in the industry to forge and build application platforms, not least of which is the Microsoft platform, which has successfully created a parallel ecosystem focused on Microsoft's Windows operating system and its derivatives for various form factors. The Microsoft platform is less democratically evolved and has succeeded in attracting developers willing to trade off choice and deployment flexibility or portability with productivity and time-to-market gains. Another broader and less homogenous ecosystem has grown around a more fluid Web and scripting language architecture, which encompasses ever richer JavaScript/HTML browser applications backed by Web servers running any of a number of dynamically typed scripting languages such as PHP, Perl, Ruby, or even JavaScript on the server. In some cases, such server languages have been implemented on Java itself, creating synergy and overlap with the Java ecosystem, but generally such developers do not consider themselves Java platform developers.

The high level of democracy in the JCP has cost Java its ability to keep up with the latest and greatest in the world of computing, where new languages and frameworks are constantly invented and proposed as new and improved mousetraps. However, the JCP has also provided the kind of platform stability needed for the broad enterprise adoption that Java has enjoyed. Oracle is unlikely to do away with this process since promulgating Java as a standard is a key value proposition of Oracle's marketing message around the company's products. Oracle is thus resigned to navigating this compromise as it whittles away at the bureaucratic excesses of the JCP in order to move it faster.

---

## **Finally Shipping Java 7**

Fully aware of the JCP issue, Oracle took decisive steps to move Java along in a faster and more incremental fashion after the acquisition. Sun released Java Standard Edition 6 at the end of 2006, bringing a number of modest improvements and changes to the Java 5 platform. The holding pattern that ensued around Sun during its sale negotiations and acquisition approval was not helpful in evolving Java rapidly, and once the dust settled after Oracle took ownership in January 2010, the Java platform improvements promised by Sun were divided by Oracle into two waves of features, namely Java 7 and Java 8. In fact, the implementation was already afoot for Java 7 even as Oracle was pushing the more incremental approach through the JCP. From a technical perspective, Java 7 brings a laundry list of improvements in multiple areas including networking, security, performance, and internationalization, but the four most significant areas of new investment are:

- ☒ **Dynamic Language Support (JSR 292).** The addition of the InvokeDynamic bytecode primitive to improve support for dynamic languages such as Ruby, Python, and JavaScript, which have captured some mindshare in recent years (The new capability allows such languages to leverage the JVM with a higher level of efficiency and performance and may drive many developers in these languages toward the larger multilingual Java ecosystem.)
- ☒ **New File IO (JSR 203).** A new system and API set for handling asynchronous file input and output known as NIO (The new APIs allow access to a wider array of file metadata, offer more information when errors occur, and support custom file systems such as emerging in cloud and Big Data environments.)
- ☒ **Explicit Multicore Support (JSR 166).** A new Fork/Join API Framework that allows developers to create explicitly parallel programs that take advantage of the increased core densities in today's microprocessors
- ☒ **Project Coin (JSR 334).** A collection of language enhancements aimed at bringing the Java syntax up to date, simplifying common programming tasks (Notable new syntax additions in Java 7 are the new switch statement, which allows more efficient code to be generated by the compiler [than a sequence of if-then-else statements], and the try-with-resources statement, which declares objects that must be closed no matter how the program terminates.)

That Java 7 shipped in mid-2011 despite internal team reorganizations related to the merger in the first half of 2010 is evidence of Oracle's purposeful management of Java. Thus, while the unprecedented four and a half year interval between Java 6 and Java 7 can be seen as a deceleration in the energy behind Java, Oracle should be recognized for breaking the logjam and moving the process along.

---

## Java EE and the Shift to Lightweight

Java Enterprise Edition is a platform that extends Java with a framework of services intended for industrial-strength security, distributed computing, and data access capabilities traditionally demanded by for large-scale enterprise applications. Java EE was first introduced in 1998 as the Java Professional Edition add-on framework building on the Java SE runtime. Java EE set out to emulate heavy-duty architectures for handling enterprise requirements, such as CORBA, and has gone through the improvements of five major releases, which built up its complexity and then started to unwind some of it in the last two releases.

Java EE has been widely adopted for building applications that require some of its included capabilities, but rarely requiring them all in a single application. Java EE was criticized from its early days for its complexity, expansive scope, memory and disk footprints, and speed of start-up and deployment. Java EE was challenged soon after introduction by lighter-weight approaches to the problems it sought to solve, most notably by the Spring Framework released in open source by Rod Johnson, founder of SpringSource (now the application platform division of VMware). Java EE has also been under challenge by lighter-weight partial implementations of Java application servers (e.g., Apache Tomcat) that focused only on the Web and Servlet part of the Java EE specification. Light container frameworks and partial Java EE implementations in fact served to broaden the adoption of Java in the enterprise — stimulating lighter-weight approaches and new innovations to be added to Java EE in later releases (e.g., Inversion of Control support in Java EE 5 and limited modularity through the Web profile in Java EE 6).

Overall, these challenges to Java EE have served to strengthen the overall Java ecosystem, even though the growth in adoption of the heavyweight (e.g., transaction support) capabilities of Java EE application servers slowed in recent years. While some of this slowdown in adoption can be attributed to open source application server challengers, a shift in the nature of applications developed in the enterprise, including expanded attention paid to user-facing application front ends, and utilizing a broader range of Web technologies has also been a significant contributor.

## FUTURE OUTLOOK

The application development landscape is in the midst of massive architectural transformations resulting from four key ongoing changes in computing:

- ☒ **Mobile.** The proliferation of mobile device form factors (smartphones, tablets, PCs, TVs, set-top boxes, etc.), which is bringing new device interaction paradigms involving touch, gesture, and speech interfaces, as well as new styles of applications with new attendant distribution and monetization modalities
- ☒ **Cloud.** The emergence of cloud computing architectures and cloud application platforms that reduce the friction in deploying back-end applications and create new scalable sets of services around which new applications will be designed
- ☒ **Social.** The dominance of social networking platforms and social features in consumer computing, creating new Internet services and new ways to monetize them through collected data (Social capabilities are increasingly making their way into enterprises both as new applications and as added features of existing ones.)
- ☒ **Big Data.** The proliferation of data and the need to collect it, store it, visualize it, analyze it, and distill it and support manual or automated decisions based on the analysis (Data is being generated by the proliferation of devices and device sensors and by the escalating density of user interaction with these devices and their supporting back-end systems.)

The pace of change requires an aggressive strategy to evolve Java if it is to remain as important over the next decade as it has been over the prior decade. It should be noted that Java is already a player in many of these domains — Java skills are employed in evolving the Android platform and ecosystem, Java is offered by many platform-as-a-service players, many social applications are written in Java, and one of the key Big Data technologies, Hadoop, is written in Java. Nevertheless, the pace of evolution of the previously mentioned technologies is relentless and accelerating. Evolving Java to competently handle the changes outlined is a tall order requiring massive investment by commercial vendors as well as significant energy from the Java community. Oracle must pursue a vision of Java that will keep it relevant in this more complex and diverse application landscape.

---

## SE 8: Parallelism and Convergence

A true test for Oracle will be the delivery of Java 8, now targeted for mid-2013. Even if it ships roughly on time, the bulk of Java 8's features would have been awaiting implementation for almost seven years. Java 8 is in the final stages of definition, with a timetable for its milestones that builds up to a production date in late 2013. While Java 8 will bring many new enhancements, the most anticipated feature areas on which the release focuses are modularity, lambda functional syntax, and JVM convergence:

- ☒ **Functional programming of Project Lambda.** Java 8 introduces functional programming constructs in the form of the changes referenced as Project Lambda. Functional programming is characterized by a declarative style of computation, which is described with lambda functions, also known as closures. Computation expressed functionally rather than explicitly through state transformation instructions permits the underlying system more flexibility in exploiting parallel processors. The Lambda language changes will also permit the addition of bulk operations for handling large data sets known as filter/map/reduce for Java. The need for better parallel support comes as Moore's law takes us into ever greater transistor densities that challenge developers' ability to utilize them with traditional serial programming. Providing functional programming capabilities in Java will permit developers to unleash the computing capabilities of new hardware architectures.
  
- ☒ **JVM convergence.** Prior to acquiring Sun, Oracle acquired BEA and inherited the JRockit JVM, which is the JVM for Oracle's strategic middleware. With the Sun acquisition, Oracle also inherited HotSpot, a more broadly adopted JVM. Java 8 promises a converged JVM that integrates the best features from both JRockit JVM and HotSpot. For example, JRockit's serviceability features such as Mission Control and Flight Recorder will be features of the Java 8 JVM. Oracle has also indicated that it will converge some of the multitude of Sun implementations so that it can focus on delivering a broader set of capabilities for fewer implementations. JVM convergence is a tricky business because the performance, reliability, and security of Java rest on the refinements of its JVM.

---

## **SE 9: Delayed Modularity**

One significant improvement to Java previously targeted for SE 8 is a new module system. Oracle recently decided that the work along this project has not progressed fast enough to make the SE 8 wave of changes, which will instead ship in SE 9.

Modularity refers to the capability of factoring out parts of the Java runtime in applications. Modularity is an ideal in software development that software architects aspire to. A large body of code such as the JDK, which grew organically for 15 years, is extremely complex and interconnected. The task of organizing Java's system libraries into groups of disentangled modules is an ambitious undertaking first mapped out in 2008 with Project Jigsaw. Delivering a more modular architecture for Java will reduce software complexity and allow separate profiles of Java to offer better compatibility across form factors.

The shifting of Project Jigsaw to SE 9 was the right decision, and Oracle was right to prioritize schedule over features. Different developers want different features in the platform, but all agree that a stable release must be delivered. The majority of the Java ecosystem cannot absorb extremely rapid changes, but the language must evolve rapidly enough to stimulate innovation. The maturity of any development process is measured by the predictability of its schedule. Modularity is a hard problem, and the benefits are indirect but potentially enormous in terms of expanding the Java ecosystem and putting Java in ever more diverse settings. But Java does not exist in a vacuum, and more lightweight technologies are constantly challenging Java as alternatives. In the era of cloud services, social interaction, and mobile app stores, a faster cadence is needed, and the two-year cycle should give way to a more incremental and faster approach to development everywhere.

---

## **EE 7: Cloudy with a Chance of Multitenancy**

Java Enterprise Edition has historically shipped on a three-year cycle. Java EE 6 shipped in December 2009, a month prior to the closing of the acquisition. Java EE 7 is currently in its late stages and is planned for release by the end of 2012. EE 7 includes a number of enhancement and changes, but one key focus has been on support for cloud computing.

Server computing is evolving to cloud computing, which relies on highly virtualized and granular resources available to developers through a high degree of automated provisioning and management. Cloud computing has been an accompanying back-end technology to the online, social, and mobile revolutions by providing the opportunistic back ends to many consumer apps. Cloud application platforms are proliferating today, but their use has been largely in the domain of new age Web, social, and mobile application back ends. Enterprises have not yet mobilized to leverage cloud platforms at scale. Java EE 7 is being planned to support this upcoming change in preparation for a wave of enterprise adoption for cloud application platforms expected to take place in the next five years.

One of the anchor capabilities that is being driven into the Java EE 7 platform is an inherent notion of workload isolation to support multitenancy. While it is possible to achieve isolation for Java applications through a virtualized instance of the JVM with its loaded Java EE classes, this is widely considered too coarse and resource extensive a solution for emerging cloud applications, which might at times call for highly granular resources. Much of the PaaS phenomenon is about supporting thousands of developers in freemium business models where costs can add up unless multiple apps are designed to run safely in a single virtual machine. Multitenancy in Java EE is intended to enable this scenario to support the needed density for industrial-scale operation of Java cloud providers.

Supporting multitenancy in a reliable and stable fashion is a challenging undertaking, and delivering this functionality is important. There is some concern in the community that the excessive focus on cloud features in EE 7 comes at the expense of advancing other aspects of the platform. IDC believes that absorption of new EE releases is very slow in the enterprise, with many organizations yet to take advantage of EE 6. While Java EE must continue to evolve on a rapid cadence, taking the time to implement cloud computing features is appropriate as a focus for EE 7.

---

## **No IDE Redux**

With the Sun acquisition, Oracle inherited the relatively popular NetBeans IDE, which IDC regularly finds is second only to Eclipse in popularity among Java developers. As an open source, general-purpose Java IDE, NetBeans is more popular than Oracle's JDeveloper IDE, which continues to be the primary IDE for Oracle Fusion Applications and Oracle Fusion Middleware. Oracle was savvy to continue its investment in NetBeans and even build on its popularity and increase its adoption. NetBeans has received support for popular scripting languages like JavaScript and PHP as well as for enterprise mobility capabilities.

Oracle also supports Eclipse and offers the Oracle Enterprise Pack for Eclipse to ensure that the sizable Eclipse developer community can work productively with the Oracle stack. It should also be pointed out that Oracle has an effort to support Microsoft ecosystem developers working with the Oracle Database, namely Oracle Developer Tools for Visual Studio, which is an add-on to Microsoft's Visual Studio, which allows developers to interact with an Oracle Database in a very similar way that Visual Studio interacts with the MS SQL Server database.

IDEs have long ceased to be the center of monetization of their own research and development, relying instead on the runtime aspects of the platform to be funded. Yet, while funding multiple IDEs can over time be costly to Oracle, a significant amount of goodwill is garnered from developers who have invested skills in learning IDEs, justifying this funding.

---

## **The Android Effect**

A Linux-derived operating system called Android has begun to challenge Java for mobile and embedded applications. To achieve maximum traction for its new smartphone platform, Google leveraged the Java developer ecosystem by creating a platform that can be programmed in the Java language. Google did not license Java from Oracle and did not use the term *Java* in its description of the Android platform. Instead, Google claimed to use a clean-room implementation of a new virtual machine, which goes by the name of Dalvik, that executes a different bytecode specification generated from the Java language code application developers write. That Java is familiar to millions of developers was likely the largest factor in its selection for Android.

In the process of defining Android, Google decided to implement many of the APIs differently so that Android applications are not compatible with Java. Many saw Android as a positive evolution for the Java ecosystem as it allowed Java developers to engage in the blossoming smartphone app industry. However, Android chips at Java's message of compatibility and creates a splinter likely to grow over time into other areas of use. Android has already spread into tablets, and it is possible that full Android PCs will one day emerge to compete with Windows PC, leveraging the Android app ecosystem as a seed portfolio. While Android may be seen by many as a net asset to the Java ecosystem, it presents a risk of fragmenting Java. Android differs from Java in two material ways. One is that it is more autocratically governed by Google, which keeps the code's early milestones in closed source. The other is that it is offered as free and open source to hardware makers, which has been a key factor in its rapid adoption. One threat facing Java is that of a parallel and competitive Android-based set of frameworks that ultimately competes with Java outside of mobile form factors, such as on ARM-based servers. Much of how the Java versus Android landscape evolves depends on the actions Google and Oracle take in the next two years as a result of competitive pressures around the Java ecosystem.

## **ESSENTIAL GUIDANCE**

Java has spurred an economy of massive proportions, which, while never explicitly sized by IDC, we believe to be well in excess of a hundred billion dollars in hardware, software, and services annually. The value of an ecosystem is reflected in the richness of the available products and services, which, once they reach a critical mass of volume and density, can lead to the long-term stability and safety that are much coveted by users of the technology. For Java, developers feel safe to invest in learning and building skills around Java and its ecosystem, precisely because of the scale of Java's ecosystem. Sustaining the long-term health of a software ecosystem in the face of a fast-changing technology landscape is a challenging job that falls to the ecosystem's leader. In Java's case, this is now Oracle.

## LEARN MORE

---

### Related Research

- ☒ *Worldwide Development Languages, Environments, and Tools 2011 Vendor Shares* (IDC #235129, June 2012)
- ☒ *Worldwide Enterprise Portals 2011 Vendor Shares* (IDC #235128, May 2012)
- ☒ *Going Mobile: The Coming Convergence of Front-End Application Platforms and Ecosystems* (IDC #234000, March 2012)
- ☒ *Application Development Trends: Developer Ecosystems in Flux* (IDC #DR2012\_BB1\_AH, Directions 12 Presentation, March 2012)
- ☒ *Windows 8 on ARM: Important New Disclosures* (IDC #lcUS23318012, February 2012)
- ☒ *IDC Predictions 2012: Application Development & Deployment* (IDC #WC20120117, January 2012)
- ☒ *Windows 8 Application Development — A Tale of Two Experiences* (IDC #230687, September 2011)
- ☒ *Worldwide Development Languages, Environments, and Tools 2010 Vendor Shares* (IDC #229273, July 2011)
- ☒ *Getting Old Apps to Windows on ARM: Not So Easy* (IDC #lcUS22873911, June 2011)
- ☒ *Worldwide Application Development Software 2011–2015 Forecast: Leveraging the Disruption* (IDC #229150, June 2011)
- ☒ *Worldwide Web Design and Development Tools 2010 Vendor Shares* (IDC #228971, June 2011)
- ☒ *Worldwide Application Development and Deployment 2011 Top 10 Predictions* (IDC #227110, March 2011)
- ☒ *Microsoft Gets Its Mobile Mojo Back with Windows Phone 7* (IDC #lcUS22642610, December 2010)

---

## **Synopsis**

This IDC study assesses the state of Java two and a half years after the acquisition of Sun Microsystems by Oracle Corp.

"The measure of success for any software platform is its ability to feed an ecosystem of developers, vendors, and technologies, working both collaboratively and competitively to build added value," says Al Hilwa, program director of Application Development Software research at IDC. "Java has been successful in building one of the largest ecosystems known in modern software."

---

## **Copyright Notice**

This IDC research document was published as part of an IDC continuous intelligence service, providing written research, analyst interactions, telebriefings, and conferences. Visit [www.idc.com](http://www.idc.com) to learn more about IDC subscription and consulting services. To view a list of IDC offices worldwide, visit [www.idc.com/offices](http://www.idc.com/offices). Please contact the IDC Hotline at 800.343.4952, ext. 7988 (or +1.508.988.7988) or [sales@idc.com](mailto:sales@idc.com) for information on applying the price of this document toward the purchase of an IDC service or for information on additional copies or Web rights.

Copyright 2012 IDC. Reproduction is forbidden unless authorized. All rights reserved.