

> SaaS ARCHITECTURE

TABLE OF CONTENTS

1.0 Introduction	1
2.0 SaaS Evolution	1
2.1 SaaS Definition	1
2.2 SaaS Four Waves Evolution Model	2
3.0 Distributed Application Architecture	4
3.1 User Tier	5
3.2 Application Tier.	5
3.3 Data Access Tier	6
3.4 Data Storage Tier	6
4.0 SaaS Architecture	6
4.1 Distribution Tier	7
4.2 Other Application Tier Components	8
4.3 Administration	9
4.4 Infrastructure Monitoring	11
4.5 Configuration	12
5.0 SaaS Reference Architecture Components	13
5.1 SaaS Architecture	13
5.2 SaaS Operations	15
6.0 Summary	15
7.0 References	16
8.0 Glossary of Terms	17

1.0 INTRODUCTION

Software-as-a-Service (SaaS) is emerging as a dominant approach to delivering software. In addition to being an application delivery model, it is also a business model. Accordingly, SaaS encompasses a broad spectrum of business, marketing and technical opportunities, issues and challenges. This paper focuses on the technical aspects of SaaS and provides an overview of the technical architectural components.

We begin this paper with a discussion of the evolution of SaaS, including a foundational definition and an overview of Saugatuck Technology's Four Waves evolutionary concept. We then cover the main components of a distributed application architecture followed by extensions to this architecture that define a more comprehensive and mature SaaS implementation. Finally, the paper presents a series of SaaS reference architecture concepts. These components are explored in more detail in a series of supporting papers.

2.0 SaaS EVOLUTION

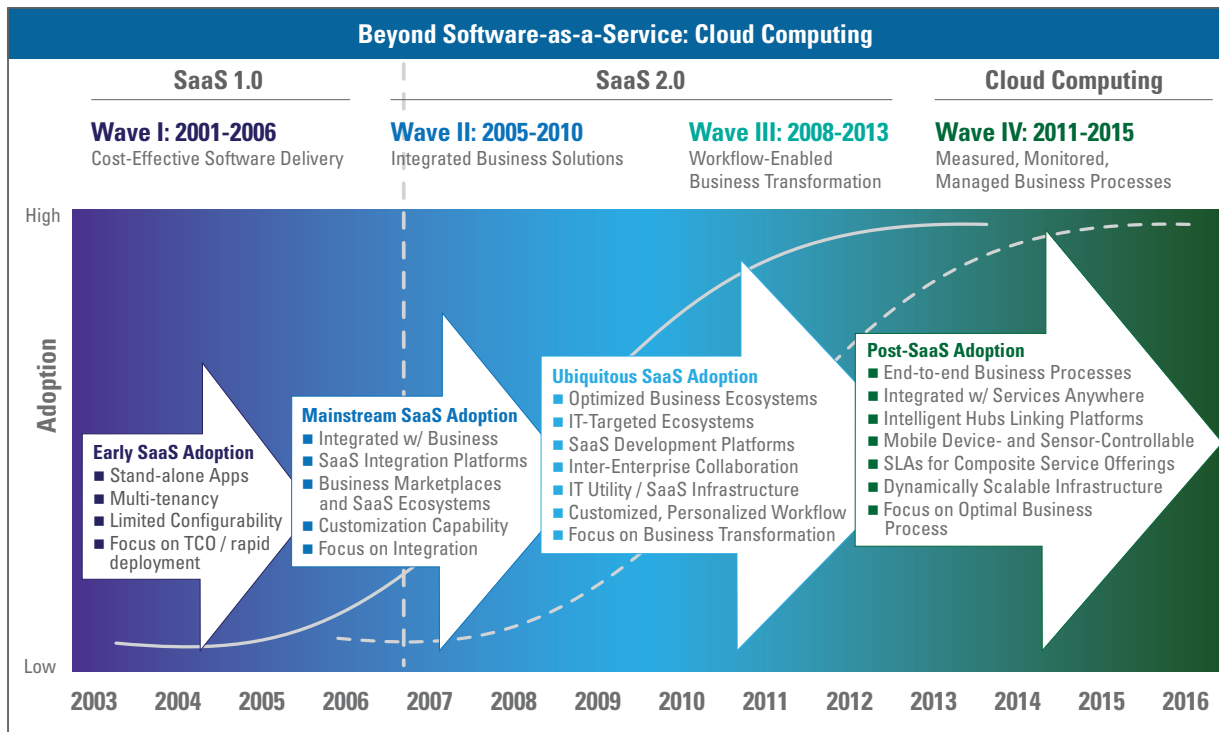
2.1 SaaS DEFINITION

Software-as-a-Service (SaaS) shares the distinction of being both a business model and an application delivery model. SaaS enables customers to utilize an application on a pay-as-you-go basis and eliminates the need to install and run the application on the customer's own hardware. Customers generally access the application via a Web browser or thin client over the Internet. SaaS is most often subscription-based and all ongoing support, maintenance, and upgrades are provided by the software vendor as part of the service. Application customization capabilities, if available at all, are generally provided to all customers in a consistent manner. From the perspective of the software vendor, the SaaS model provides stronger protection of its intellectual property, operational control of the environment running the software, and generally a repeatable revenue stream from the service subscription fees. Software vendors have varying capabilities and applications can come in varying flavors but SaaS applications most typically support many unique customers using a single instance of that application—also known as multi-tenancy.

2.2 SaaS FOUR WAVES EVOLUTION MODEL

In a 2007 Strategic Report¹, Saugatuck Technology defined the Three Waves theory of evolution of the SaaS market. In a recent Saugatuck research paper, they have extended this to include a fourth wave². The Four Waves concept provides a useful approach to understanding the emerging and maturing SaaS model and can be helpful to a software vendor looking to move an existing ASP-based (application service provider) or client-server application to one that is SaaS-enabled. Figure 1 shows a diagram of the four waves from the latest Saugatuck paper.

In the first wave, SaaS vendors provide very simple offerings usually involving stand-alone applications (generally either entry-level or function-specific) and the focus is on cost savings due to ease of deployment, maintenance and use. A recent article from SoftwareCEO³ noted an IDC “Worldwide Black Book” finding that only 20% of IT budgets are going toward software while the rest goes toward the people, services and hardware needed to run this software. Since it is the software that corporate end users actually want and not all the overhead associated with it, there is a serious mismatch of cost versus value delivered. The first wave therefore addresses this basic value proposition.



In the second wave, the tail end of which is where we find ourselves today, there is more mainstream SaaS adoption and the solutions are more sophisticated. Here we see the emergence of more customization capabilities and a big focus on integration. As SaaS takes on a bigger role in the overall corporate ecosystem, the ability to integrate these solutions as part of the overall corporate data flow becomes important.

In the third wave, SaaS evolves, matures and gains more widespread acceptance to the point where it becomes a critical part of the overall enterprise software strategy. As a result, it becomes a facilitator of business transformation. Integration between SaaS business services and on-premise services, as well as inter-company sharing of data, is a major feature of this wave. An excellent example is the integration of a corporate human resources application such as Oracle PeopleSoft with a SaaS CRM solution such as Salesforce.com. This reliance on SaaS solutions integrated with the fabric of existing corporate data sources therefore enables transformation of the business itself, making existing processes more efficient and cost effective and in some cases creating new business processes that heretofore were not feasible.

Figure 1:

Saugatuck SaaS Wave

Evolution Model

Source: Saugatuck Technology Inc.

The fourth wave addresses the emergence of the cloud computing paradigm and speculates that leveraging these new environments “is the natural progression for SaaS, the IT utility concept, and business process outsourcing and transformation.” Cloud computing offers on-demand infrastructure and combined with on-demand software (SaaS) can provide more flexibly deployed and managed solutions, serving to accelerate the business transformation that marked the third wave.

3.0 DISTRIBUTED APPLICATION ARCHITECTURE

When the use of the Web for business applications began to catch on in late 1995, a distributed application architecture (then sometimes referred to as an N-tiered architecture) emerged. This looked different from the traditional client-server architecture where the database generally sat on a centralized server and the thick client containing the user interface and application logic was installed on the desktop of each user. There were various flaws in this model, including scalability issues due to the dedicated database connections from each user as well as the operational overhead of the distribution, installation and maintenance of the client software on users’ desktops.

The distributed application architecture was really made possible by two things. First, the Web browser emerged as a ubiquitous tool on the desktop. Second, a class of software known as an application server emerged that allowed rich application functionality to be delivered to the desktop via the Web browser. Gone were the distribution and maintenance costs as the client (Web browser) was now either installed within the operating system, or at least freely available for simple download and install. Also gone were the scalability issues because the application server environment split up the various components handling user requests so they could be scaled as needed.

Figure 2 shows the various tiers of this architecture.

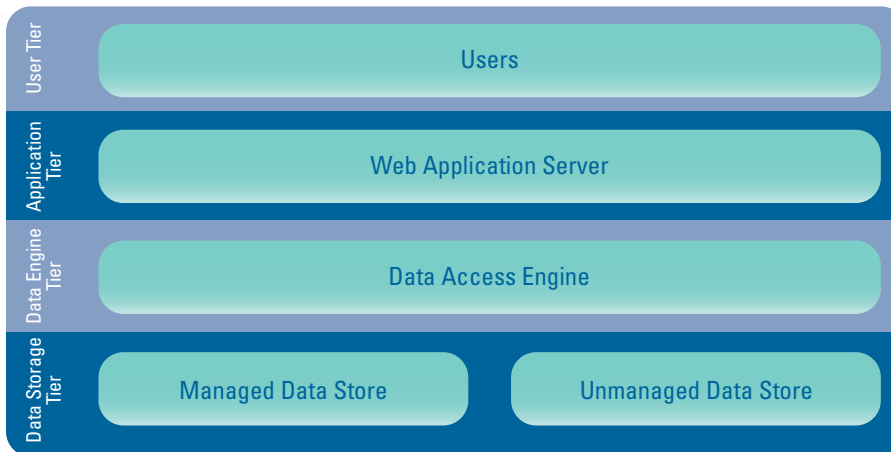


Figure 2:
Distributed Application Architecture

3.1 USER TIER

The user tier usually consisted of the Web browser that already existed on the desktop or could be easily downloaded and installed for free. Alternates to the Web browser could include thin client applications. These usually only handled the input and output from the server (where most of the processing and the business logic resided), and could often be downloaded as and when needed.

3.2 APPLICATION TIER

The application tier included the application server which handled HTTP/HTTPS requests from users and processed those requests with business logic and database access as required via the data engine tier. The application tier generally provided such application services as clustering of application servers and load balancing between them for scalability and resilience, management of user session data, management of database connection pools and management of application objects containing business logic.

3.3 DATA ACCESS TIER

The data access tier handled the processing of all requests for data, either from a structured and managed SQL data store or from an unstructured and unmanaged data store such as a file system. This tier generally provided clustering for scalability and resilience. In addition to providing a layer of abstraction, separating business logic from data, there were other reasons for physically separating this tier from the application tier. These included different hardware requirements (i.e., optimized disk access for a database server versus additional RAM needed by the application server) and added security (segregating this tier to an internal network segment to reduce the risk of unauthorized outside access to critical data).

3.4 DATA STORAGE TIER

The data storage tier was used for the physical storage of data. In some cases this was the same physical server as the data engine tier. In others, a Storage Area Network (SAN) was used for storage optimization and resilience.

4.0 SaaS ARCHITECTURE

The SaaS model encompasses the concepts in the distributed application architecture but further extends the architecture to include components to facilitate and enhance the business model. A traditional software vendor is primarily concerned with application functionality and their customers are responsible for operating and managing the respective environments in which they run the software. A SaaS vendor, on the other hand, is equally concerned with operating and managing the environment that supports all their customers.

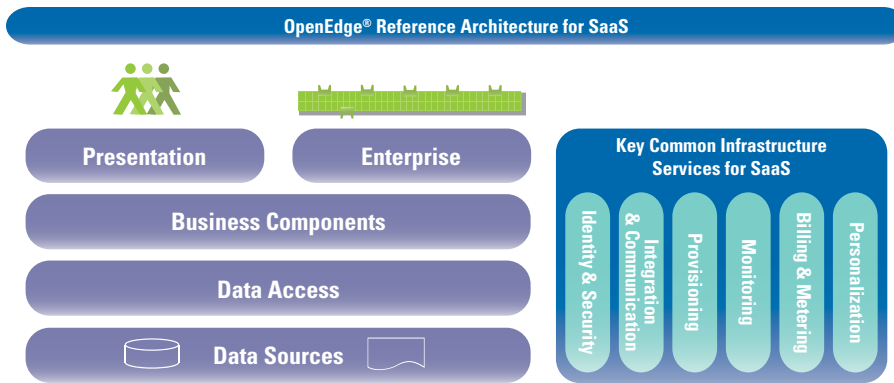


Figure 3:
SaaS Architecture

Figure 3 shows a conceptual diagram of this more comprehensive architecture. An additional distribution tier is added to acknowledge that service requests may need to be routed among more than one physical operating environment for various reasons. Additional application tier components have been identified that take into account the additional functionality a mature SaaS offering may require. The issues of administration and monitoring are presented as components that operate across tiers. These components are critical to the efficient operation of a SaaS business. Finally, the topic of configuration is covered so that configurable application functionality can be considered when appropriate.

4.1 DISTRIBUTION TIER

The distribution tier is added to the SaaS architecture model to handle load balancing. This term is generally associated with techniques for spreading work to optimize resource utilization, throughput or response time. With a SaaS business this is certainly appropriate to consider so that your customers can be provided with the best level of service possible. However, there are other reasons why service requests may need to be routed to different physical operating environments.

One reason is to leverage multiple, geographically disparate data centers for resiliency. This topic is covered in more detail in subsequent papers on hosting and replication, but the idea is that different customers can be routed to different primary sites. All sites are used for production service requests and they act as alternate sites for each other, thereby efficiently utilizing resources and providing high availability.

Another reason for routing certain customer service requests to specific environments may be to comply with international data privacy laws. This topic is complex and worthy of its own technical paper. The fact is that data privacy is a major issue and different countries have taken very different approaches to protecting their citizens' data. In spite of efforts, such as the SafeHarbor certification program developed by the U.S. Department of Commerce to facilitate compliance with European Union data privacy laws, certain customers from certain countries may require that their corporate data be physically stored in their country. If the SaaS vendor wants to develop business in such countries, internationally disparate production environments may be required and routing of requests will be necessary.

4.2 OTHER APPLICATION TIER COMPONENTS

The SaaS architecture model adds some granularity at the application tier. Rather than just a general application server running your application, there are different types of specialized servers providing specific types of functionality.

4.2.1 Identity Management Server

Security is an important issue for a SaaS environment. Having the ability to flexibly handle identity management in a standardized way will go a long way toward allowing customers to leverage multiple modes of authentication. An example of an identity management server is OpenSSO⁵. This is an open source access management and federation server platform that is based on the Sun Java System Access Manager and will be the core of upcoming releases of Sun's Access Manager and Federation Manager. More details on identity management are provided in a supporting technical paper on Security and Privacy.

4.2.2 Integration Server

As discussed in the Four Waves evolution model (in Section 2.2), integration is becoming a key issue as SaaS offerings mature. Providing a specialized integration server with existing ties to industry standard platforms

such as HR, CRM, financial, etc., will make it much easier for customers to integrate a SaaS solution with their existing systems. Integration is covered in more detail in a separate technical paper.

4.2.3 Communication Server

Both outbound and inbound communications are additional components of a SaaS offering that are often required, yet sometimes overlooked. At a minimum, it is useful to have the ability to reach users in multiple ways (i.e., email and text messages) to let them know they have work to do in the SaaS application. An email server, referred to as a mail transport agent or MTA, is a simple process to run, but providing a robust email infrastructure requires more thought and design. Also, SMTP email is not necessarily the only communication protocol that maybe supported, so thinking of a communication server as a more generalized component in the architecture is helpful.

4.3 ADMINISTRATION

Administration of the overall SaaS environment and business itself is a unique component of a SaaS solution. Since SaaS is about delivering a service, considering administration of the business as a component within the overall architecture makes sense. Billing, operations and support aspects of this component are briefly discussed below but the topics are more fully addressed in separate papers including Service Provisioning, Billing and Metering, and Subscriber Management/Self Service.

4.3.1 Metering

Depending on the billing model for a SaaS solution, there may be a need to meter customer usage and bill accordingly. This metering can either be provided as a component of the SaaS solution itself or may be a service supplied by a hosting provider. Regardless of how it is implemented, the metering solution will need to pass its resulting usage data on to the billing system that actually generates and presents customer invoices for payment.

4.3.2 Billing

Billing can also be handled by a component within the SaaS environment or by a third party billing service. The billing process either accepts usage input from a metering component or contractual details in the case of a fixed recurring fee. A periodic invoice is created from this data and presented to the customer for payment.

4.3.3 Payments

Once an invoice has been generated and presented to the customer, payments must be processed. This can include processing credit card transactions, electronic funds transfers or receiving and depositing physical checks. This part of the process is also a candidate for outsourcing and there are many companies that provide this service.

One consideration for payments in the SaaS model is that a service, once consumed, usually cannot be returned. Therefore, the service should ideally be paid for in advance of use. This is not always possible, especially if billing is based on metered usage, but it will be useful for a SaaS vendor to consider its policies surrounding payments and service cutoff in the event of non-payment.

4.3.4 Operations

The operations group is generally responsible for keeping the SaaS environment up and running and for making sure the service is provided for each of the customers. If the administration component has the management tools to assist the operations group, it keeps everything streamlined. For example, these can include tracking contractual details for each customer, customer contact information, contract expiration dates, disk usage per customer, users per customer, etc.

4.3.5 Support

Ideally, the support, help desk or ticketing system is integrated into the administration component. If the ticketing system is then integrated into the SaaS application itself, clients can submit tickets either directly from the application or by phoning the help desk. In addition, they will also be able to keep track of ticket status.

4.4 INFRASTRUCTURE MONITORING

A SaaS infrastructure needs to be both locally resilient (basic hardware redundancy and local failover) as well as geographically resilient (DR failover capability to a remote location in the event of a complete data center or regional outage). It is important to be able to operationally monitor the entire environment from the hardware level all the way up to the application internals so that outages can be avoided or handled promptly if they do occur. There are many tools that can provide this type of monitoring but it is also important to also consider Service level Agreement (SLA) reporting, support metrics and quality metrics. These can be provided by a robust monitoring component.

4.4.1 SLA Reporting

Today, SLAs are becoming more and more a standard part of a SaaS contract. One of the easiest ways to manage SLAs is to proactively monitor and report on the appropriate metrics. Automating this process and providing access to the reports right from the application gives customers confidence in the stability of the SaaS environment.

4.4.2 Support Metrics

Tracking support metrics is an important part of managing the overall support function. Understanding how many support calls are being handled in total, and per client, how many tickets are entered, the average time they stay open and correlating support calls to environment outages are all metrics that help a SaaS business to manage how well their support staff is operating and how they will need to scale staff as the business grows. Much of this data can be provided by a monitoring component.

4.4.3 Quality Metrics

Measuring quality can be a subjective exercise but there is value in defining and institutionalizing quality metrics. The feedback loop created by this process can be incredibly helpful in making sure a business is performing at the desired level. A well designed monitoring component can easily be a source for such metrics. Building functionality to capture and report such quality data should be considered.

4.5 CONFIGURATION

As SaaS solutions mature through the Four Waves Evolution Model, (outlined in Section 2.2) they are marked by increasing levels of configurability and customization.

4.5.1 Functionality

There are often cases where different functional approaches can work for different customers within the same application. One example maybe a hierarchical data structure that may generally be best viewed in a user interface as a tree. However, certain customers may have so much data that it may be more effective to display in a paginated table interface. In a SaaS solution, there is no reason why both types of functionality cannot be implemented and then allow the customer to configure the application to behave in the way that works best for them. Thinking about how to implement configurable functionality rather than different customized versions of the application is critical to achieving the goal of greater maturity in a SaaS implementation.

4.5.2 Look and Feel

With today's standardized approach to controlling the look and feel of a Web-based user interface via technology such as cascading style sheets (CSS), it is strongly recommended to develop a SaaS application so this look and feel can be customized by each customer. This can include changing fonts, colors, etc., and adding a customer logo. Sometimes this is referred to as skinning an application and software that allows a skin to be applied is called "skinnable."

4.5.3 Customization

The SaaS model generally calls for all customers to use a single version of the application as opposed to separate customized versions. A challenge for SaaS vendors is therefore how to provide their customers with the ability to customize the application to suite their individual needs. One approach to this is to allow for the extension of system objects via custom

fields. If implemented properly by using name-value pairs, these custom fields can be easily defined and utilized without the need to change database schema. This topic is covered in more detail in a separate paper.

5.0 SaaS REFERENCE ARCHITECTURE COMPONENTS

This paper presents an overall SaaS reference architecture for consideration. The topics outlined below are addressed in more detail in several supporting technical papers. These topics are divided into two major categories: SaaS Architecture and SaaS Operations.

5.1 SaaS ARCHITECTURE

5.1.1 Application Tenancy

The dictionary definition of tenancy has a strong connotation of occupancy. The concept of multi-tenancy, or the occupation of a SaaS environment by multiple tenants, is a central concept in the definition of SaaS presented in this paper. This topic will be expanded in another paper in this series to include alternate implementation options.

5.1.2 Security and Privacy

Concerns over security and privacy of corporate data in an off-premise, third-party environment were a major impediment to the early adoption of SaaS. Now that strong standards have emerged around these issues, it is easier for customers to do the necessary due diligence to make sure these standards and controls are in place. These standards and the associated controls are discussed in depth in another paper in this series.

5.1.3 Application Configuration

We briefly touched on application configuration in this paper. However, this topic is developed in more depth to cover things such as configuration of security policies, output formatting and default permissioning (authorization).

5.1.4 Customization and Personalization

Customization and personalization can be thought of as related to configuration. However, there is enough detail to be discussed around these issues to warrant a separate paper.

5.1.5 Integration

Integration is presented as one of the components in the SaaS architecture above. This topic is addressed in more detail in a separate paper along with a discussion about the many issues surrounding integration of a customer's existing systems with a SaaS offering.

5.1.6 Scalability

The fundamental SaaS concept of a single application instance supporting multiple tenants calls for scalability. This topic will be discussed in another paper as it applies to each of the tiers of the architecture as they each lend themselves to different approaches.

5.1.7 User Interface

The user interface is where the customer meets the application, so it better be good! Ideas for consideration include navigation, usability, supporting different devices and internationalization.

5.1.8 Subscriber Management/Self-Service

One of the great things about the SaaS model is that the effort to gain a client is rewarded by the opportunity to offer them as many additional services as you want. Therefore, the way in which subscriptions are managed and the extent to which customers are enabled to avail themselves of these additional services determines how well the SaaS business can scale.

5.1.9 Billing and Metering

Although we are focusing these papers on architectural aspects of a SaaS implementation, it is also important to remember that SaaS is also a business. Since the service is not offered for free, it is important to consider how billing is handled and whether or not the service requires metering as part of the pricing model.

5.1.10 Service Provisioning

The ability to quickly and easily provision service once a customer has contractually agreed to use your service is critical to success. Various aspects of service provisioning are explored for consideration in a SaaS implementation.

5.2 SaaS OPERATIONS

5.2.1 Replication and Recovery

Resilience and failover capabilities are critical for a SaaS application because all of the customers are relying on the SaaS vendor to protect their data and keep the service running. Various approaches to both topics will need to be explored.

5.2.2 Hosting

Hosting options for SaaS run the gamut from self-hosting to collocation all the way up to fully optimized platform-as-a-service offerings specifically designed to support SaaS businesses. The various options will be discussed in another paper including the latest trends in virtual cloud computing.

5.2.3 Testing QA/QC

System testing, quality assurance and quality control are all important aspects of any software development effort. Given the potential impact of poor code quality across all the tenants in a SaaS environment, it is important that these practices are given proper attention. Different systematic approaches to these topics will be discussed in another paper.

6.0 SUMMARY

The SaaS approach to deploying software solutions is emerging and evolving. As corporations continue to expand their acceptance of these solutions, as existing software vendors seek to take advantage of this model and as more new SaaS vendors emerge, it is helpful to develop an

architectural model that encompasses all of the issues a SaaS offering may need to address. This paper has presented such a model, explaining it in terms of an extension from the more traditional distributed application model.

In addition, a series of SaaS reference architecture concepts were presented in order to provide an overview of supporting technical papers that cover these topics in more detail. Together with this foundational document, these papers comprise a comprehensive roadmap and guide for SaaS Providers. Due to the inherent dynamic nature of the evolving SaaS model, this body of work represents a snapshot in time and the current state of many issues. As new technologies and standards continue to emerge, this model will surely grow and evolve too.

7.0 REFERENCES

¹Saugatuck Technologies, *2007 Strategic Research Report SSR-342*, “Three Waves of Change: SaaS Beyond the Tipping-Point,” May 3, 2007.

²Saugatuck Technologies, *2006 Strategic Research Report SSR-239*, “SaaS 2.0: Software-as-a-Service as Next-Gen Business Platform,” April 26, 2006.

³Graham, Gordon, Editor, *SoftwareCEO*, “13 habits of highly effective SaaS companies,” February 5, 2008 — <http://www.softwareceo.com/attachments/opsource/com020508.php>

⁴M. West, B. Guptill, *Saugatuck Technologies Strategic Perspectives STR-458*, “Saugatuck SaaS Research: Waves and Platforms in the Cloud,” April 29, 2008.

⁵OpenSSO, the Open Web Single Sign-On Project — <https://opensso.dev.java.net/>.

8.0 GLOSSARY OF TERMS

TERM	DESCRIPTION
ASP	Application Service Provider (ASP) is a term that describes an alternative application deployment model. Applications deployed in an ASP model are most often hosted by a service provider outside of the end customer’s premises, and accessed via client-side application or Web browser over a private connection. Traditional application licensing and maintenance conventions apply in this model.
CSS	Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in a markup language.

(Continued next page)

TERM	DESCRIPTION
Customer/Tenant	Customer/Tenant is a term in this document that refers to the company or business that subscribes to the SaaS offering.
Customization	In the context of a SaaS-enabled application, customization is defined as changes to the business logic, user interface, and/or database on behalf of a single customer, and requires application code to be recompiled. In practice, most SaaS-enabled applications either do not allow for customization, as it 'breaks' the scalability model by requiring the service provider to support multiple code bases, increasing costs, or customization provided to one customer is made available to all customers and users.
HTTP	Hypertext Transfer Protocol is a communications protocol used to transfer data on the Web.
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer is a secure HTTP connection, combining a normal HTTP interaction over an encrypted Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connection.
Multi-Tenancy	Multi-Tenancy is an architectural design concept used to maximize customer and user scalability. As well as to provide maximum operational scalability and lower support costs for the service provider by leveraging common user interface, business logic, and/or databases for all customers and users. Multi-tenancy at all layers of the application is considered an architectural ideal but not required at all layers to realize benefits. The cost savings often associated with multi-tenancy are often considered key to building and maintaining lower cost service offerings.
Personalization	Personalization in the context of a SaaS-enabled application relates to minor 'tweaks' to the configuration to support the business process of customers and users. The ability to personalize some elements of a SaaS-enabled application are most often designed into the application at the time it was built.
RAM	Random Access Memory is a type of computer data storage in which data items can be accessed in any order versus requiring the movement of the recording medium or a reading head. This generally refers to the volatile computer memory where the operating system and applications run as opposed to the non-volatile disk drives where data is stored.
SaaS	Software-as-a-Service (SaaS) shares the distinction of being both a business model and an application delivery model. SaaS enables customers to utilize an application on a pay-as-you-go basis and eliminates the need to install and run the application on the customer's own hardware. Customers generally access the application via a Web browser or thin client over the Internet. SaaS is most often subscription-based and all ongoing support, maintenance, and upgrades are provided by the software vendor as part of the service. Application customization capabilities, if available at all, are generally provided to all customers in a consistent manner. From the perspective of the software vendor, the SaaS model provides stronger protection of its intellectual property, operational control of the environment running the software, and generally a repeatable revenue stream from the service subscription fees. Software vendors have varying capabilities and applications can come in varying flavors but SaaS applications most typically support many unique customers using a single instance of that application also known as multi-tenancy.
SaaS Offering	A SaaS Offering in this document is defined as a turnkey service offering, which includes the application license, maintenance, application support, subscription pricing, and hosting and associated delivery infrastructure.
SaaS-Enabled Application	A SaaS-Enabled Application is defined as an application that has been designed and built to be deployed and consumed in a service model, and incorporates many of the attributes previously defined above.
SAN	Storage Area Network is an architecture to attach remote computer storage devices to servers in such a way that, to the operating system, the devices appear as locally attached. SANs are generally optimized for the storage of large amounts of data and contain functionality to improve resilience.
SQL	Structured Query Language is a standardized programming language for querying and modifying data and managing databases.
Users	User in this document is defined as an employee of the customer, as in the ultimate user of the service.



PROGRESS SOFTWARE

Progress Software Corporation (NASDAQ: PRGS) is a global software company that enables enterprises to be operationally responsive to changing conditions and customer interactions as they occur. Our goal is to enable our customers to capitalize on new opportunities, drive greater efficiencies, and reduce risk. Progress offers a comprehensive portfolio of best-in-class infrastructure software spanning event-driven visibility and real-time response, open integration, data access and integration, and application development and management—all supporting on-premises and SaaS/cloud deployments. Progress maximizes the benefits of operational responsiveness while minimizing IT complexity and total cost of ownership.

WORLDWIDE HEADQUARTERS

Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA
Tel: +1 781 280-4000 Fax: +1 781 280-4095 On the Web at: www.progress.com

For regional international office locations and contact information, please refer to the Web page: www.progress.com/worldwide

Progress, and Business Making Progress are trademarks or registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and other countries. Any other trademarks contained herein are the property of their respective owners. Specifications subject to change without notice.

© 2008-2009 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Rev. 11/09 | 6525-128713