Platform As A Service Development Cost & Security

Aspen Olmsted
College of Charleston
Department of Computer Science, Charleston, SC 29401
e-mail: olmsteda@cofc.edu

Abstract— **In this paper, we investigate the problem of development costs in Platform-as-a-Service (PaaS) cloud-based systems. We develop a set of tools to analyze the size of code executed to support features in the PaaS. In this research, we specifically focus on stable, open source platforms to ensure as much of an equivalent offering from each platform with a distinction made between PaaS and Platform Infrastructure as a Service (PIaaS). The focus of the paper is on the features both functional and non-functional provided to the developer that is not provided by traditional network operating systems. On the functional side, we look at features provided by the platform to assist the developer in programming tasks the software must do. On the non-functional side, we look at security defenses the platform provides to protect the end users data. Our study demonstrates a savings cost of nearly thirteen million dollars to develop the application services provided by a typical PaaS.**

Keywords-PaaS; cloud computing; CRM

I. INTRODUCTION

In this work, we investigate the problem of estimating the cost of developer services provided by a platform-as-a-service (PaaS) cloud-based system. In traditional client-server architectures, developers expend considerable effort developing functionality that is not specific to the business domain where the application will operate in. This work builds on our earlier work on development effort estimation [1].

Cloud computing has traditionally been made up of three broad categories of offerings:

- Software as a Service (SaaS) – This category includes applications that run in a Web browser and do not require any local software or hardware besides a Web browser and an Internet connection. Examples of software in this category include Google Docs [2] and Microsoft Office 365 [3].
- Infrastructure as a Service (IaaS) – This category includes virtualization software that allows an operating system to be run in the cloud. Typically, the user will pick a hardware configuration and install an operating system into the virtual hardware configuration. IaaS was designed to free the user from the purchase of hardware and allow for easy hardware upgrades. Examples of IaaS offerings are Amazon EC2 [4] and Rackspace [5].
- Platform-as-a-Service (PaaS) – This category includes pre-built components that a developer can use when developing a cloud application. The goal of PaaS is to allow the developer to focus on the development of a solution for the business functions rather than software functions that span many application domains. A good example of PaaS is force.com where the developer is provided many of the essential parts of an application out of the box.

Over the years, software development has matured to allow the developer to spend a larger percentage of their development time on the business problem instead of the infrastructure for the application. In the early days of programming, each instruction the programmer wrote matched an instruction in the hardware. The late 1980s and 90s were dominated by 3rd generation languages such as C, PASCAL, and ADA where each instruction written by the developer was compiled to many machine instructions. The first decade and a half, of the 21st century, have been dominated by bytecode compiled languages which have runtime engines that execute the code on different hardware platforms. The Java Runtime Engine (JRE) and the Microsoft .NET Runtime Engine (.NET) are the most dominant examples of the bytecode engines that free the developer from thinking about the underlying hardware. PaaS is the next evolution in freeing up the developers' times, so they can focus on the problem they are trying to solve instead of the technical plumbing required for the solution.

The organization of the paper is as follows. Section II describes the related work and the limitations of current methods. In Section III, we document typical services provided. Section IV analyzes different PaaS providers and the services they provide. Section V explores the alternative costs to develop the individual services. We give a motivating example in Section VI. In Section VII, we look at software security defenses as provided by the different platforms. We conclude and discuss future work in Section VIII.

II. RELATED WORK

The NIST (National Institute of Standards) definition of "cloud computing" defines PaaS as “the capability provided to the consumer [...] to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment” [6]. In the same document, they define SaaS and IaaS similarly to our definitions in the introduction.

Kolb and Wirtz [7] investigate ways to construct applications for the cloud that are portable across different

PaaS providers. Their work assumes lower level services compared to the offerings than our work. We are less interested in maintenance costs to move platforms as we are in startup costs for greenfield Engineering. In software engineering, greenfield engineering occurs when you are starting from scratch or are re-engineering your product on a different architectural paradigm in which you cannot port your current code base.

Baliyan and Kumar [8] explore how services provided by a PaaS provider affect the software development lifecycle (SDLC). Again, in their work, they consider just a few services. In our work, we think about many more services. The larger perspective on service would have an even greater impact on their work.

In our model of services, end users can create new objects, new forms for data entry and new reports to display the data in both detail and aggregate form as well as new dashboards. Ng [9] looks at PaaS as a model for deploying end-user programming through a model of Tasks. The programming model provided by the platforms in our study has demonstrated success in allowing end users to extend the application.

Boehm, Clark, Horowitz, Westland, Madachy, and Selby [10] developed an algorithm to estimate effort for a software engineering project. The algorithm uses variables that represent a programmer's experience and programming expertise required in the project. For this study, we used the “nominal” value for each variable to get an average cost. Madachy [11] provides an online tool to calculate the effort including maintenance over the life of the software.

III. PaaS Services

With PaaS, the developer does not need to be concerned with the operating system on which the specified platform runs. For example, the platform will provide a service to save a file, and the developer does not need to worry about what operating system the platform is running. We group the service offerings into two distinct categories:

A. *Infrastructure Services*

- Node Configuration – This service allows the end user to modify configuration settings to allow the system to scale to handle larger or smaller workloads by adding or removing nodes, storage or Central Processing Units (CPUs). This service allows the implementation to start with minimal hardware to save costs during start-up. Additional resources can then be added as the application user base grows without the need to re-engineer the application.
- Load Balancing – This service allows the end user to setup multiple systems to ensure uptime when loads are higher, or network partitions occur. Each system is an exact replica, and the load will be distributed across the replicas. The application will need to be designed properly for replication. The system must also not store resources in a specific replica as each request could be sent to a different replica. Both persisted data and session state should be stored in the database server.
- Logging – The logging service allows an audit log to be enabled to help diagnose application and platform issues. The service should allow the log to be toggled on and off so that space is not wasted when an audit is not needed. Ideally, there will be different granularity of audits available, such as errors, warnings, and information.
- Database – The database service allows the application data to persist across executions of the application. Traditionally, this has been a relational database such as Oracle [12] or MySQL [13] but may also be a NoSQL [14] database that is better at distribution. The database service should provide: create, read, update and delete (CRUD) services and potentially transaction support.
- Scheduled Jobs – This service allows bulk operations to be scheduled at specific and recurring intervals. Example jobs include sending out bulk emails, updating de-normalized database fields and communicating with external systems. Often, this service is delivered through a cronjob interface where jobs can be scheduled down to the specific second of each hour.

B. *Application Services*

- Authentication – The authentication service provides a way to define users and allow authentication in the application being developed. Ideally, this would provide both the administrative tool for creating users and groups along with the user interface with which the end users interact to authenticate themselves. The service should provide multifactor authentication which incorporates information the end user knows along with someplace they are or something they have.
- Authorization – The authorization service provides a way to define which users can see different data, forms, and reports in the application. The authorization service should provide an administrative tool to assign access permission to both users and groups to objects created in the system. The objects should be both standard objects and custom objects defined by the developer and end users.
- Rule Engine – A rule engine allows for customization of correctness rules at implementation time. Business rules control

organization policies that may change often and should not be coded in the software solution.

- Workflow – This service provides for several discrete application steps to be sequenced together. Often, a human interaction (approval) is part of the workflow.
- Bulk Email – Bulk email allows for email marketing with proper adherence to email spam rules [15]. Bulk email may be used for attracting or recruiting new customers in addition to confirming transactions with current customers.
- Importing – An importing feature allows the end user to import new instances of objects into the platform. Ideally, this would allow data from several different data formats including Comma-Separated Values (CSV) and Microsoft Excel workbook. The tool should provide a validation step so that imported data does not corrupt the current database.
- Exporting – An exporting feature allows the end user to dump instances of the objects into an external file such as a comma-separated value (CSV) or Microsoft Excel workbook. The tool should allow a query by example (QBE) where novice users can visually build export queries and see the results in the application.
- Activity tracking – Activity tracking allows for linking of phone calls, emails, meetings, and notes to objects persisted by the application. Activities may originate in an external system with an interface to the new system that is being built. An example could be a Web browser extension that allows emails in a Web email application to be linked to a related activity to an object in the new system.
- Object Customization – Object customization allows end users to add additional data to be collected in the application without changing the source code. Most enterprise systems require some form of customization either through integration to external systems or enhancements to specific features in the current system. Object customization allows the end user to make the changes without needing the software to be modified at each individual enterprise.
- New Object Creation – New object creation services allow end users to define new objects to store data that is collected in the application. Like with object customization, new object creation can be used to customize the software without changing the source code. Often, the new objects need to relate to a current object in the system. These related objects should seamlessly be displayed in the user interface.
- Detail View – The detail view renders the object details based on the configured layout. The detail view also renders one-to-many related data. The related data is often rendered in tabs.
- Edit View – The edit view renders an editable object screen based on the configured layout. The edit view is used to modify one specific object and potentially related objects.
- Data Update – The data update service provides a CRUD interface to a backend data store. The data update service abstracts the vendor specifics of the back-end data store services and allows business rule hooks to fire on the CRUD operations.

TABLE I. APPLICATION SERVICES BY PLATFORM

Service	Salesforce	Zoho	SugarCRM	SuiteCRM	vTiger	Heroku
Authentication	✓	✓	✓	✓	✓	✓
Authorization	✓	✓	✓	✓	✓	
Rule Engine	✓		✓			
Workflow	✓	✓	✓	✓	✓	
Bulk Email	✓	✓	✓	✓	✓	
Activity tracking	✓	✓	✓	✓	✓	
Object Audit	✓		✓	✓	✓	
Importing	✓	✓	✓	✓	✓	
Exporting	✓	✓	✓	✓	✓	
Object Customization	✓	✓	✓	✓	✓	
New Object Creation	✓	✓	✓	✓	✓	
User Interface Customization	✓	✓	✓	✓	✓	
Multi-Select Fields	✓	✓	✓	✓	✓	
Report Display	✓	✓	✓	✓	✓	
Report Creation	✓	✓	✓	✓	✓	
Dashboard Display	✓	✓	✓	✓	✓	
Dashboard Creation	✓	✓	✓	✓	✓	
Web-services	✓	✓	✓	✓	✓	
Mobile Application	✓	✓	✓		✓	
Partner Portal	✓					
Customer Portal	✓		✓	✓	✓	
Anonymous Sites	✓					✓
Price per user/month	$25	$35	$65	N/A	N/A	N/A

- User Interface Customization – The user interface customization service allows for forms in the application to be modified by the end users without changing the source code. This is often required to allow implementations to vary slightly by collecting custom data.
- Multi-Select Fields – Multi-select fields are a way to simplify end-user customizations. A multi-select field represents an easy way to store a one-to-many relationship of data without the need of adding new objects. Multi-select fields also save on the number of tuples stored in the system. Often, cloud providers charge for data storage based on the number of tuples [16].
- Report Display – The report display service allows execution of pre-defined reporting queries. The report display should prompt the user with replaceable run-time parameters and be exportable to PDF and spreadsheet formats. Ideally, there would be a scheduling service through which the report parameters would be based on the run date. For example, a start date parameter should be replaced based on an offset from the date the report is run.
- Report Creation – The report writer service allows both the developer and the end users to define management information system (MIS) reports that can be run and customized by the changing of run-time parameters. Typically, this includes both tabular reports that group rows of records with aggregate calculations and cross-tab reports that aggregate values based on the intersection of the row and column.
- Dashboard Display – The dashboard display service renders dashboard charts and allows them to be refreshed automatically. The dashboard is a graphical display of a metric the organization wants to measure.
- Dashboard Creation – Dashboards allow both the developer and the end users to define graphical dashboards that allow visualization of data stored in the application. Dashboards typically are bar or pie charts and are updated several times an hour.
- Mobile Application – A mobile application allows end users to perform CRUD operations on objects stored in the application without the need of creating custom mobile applications. Similar to the detail and edit view services above, any object in the system should be visible and editable.
- Partner Portal – A partner portal is a service to provide pages, forms, reports and dashboards to authenticated users with a lower training level. Typically, these are users that use the application infrequently compared to an employee.
- Customer Portal – A customer portal is a service to provide custom pages and forms to authenticated users with no training required. The service is intended for customer self-service sites where the customer can identify themselves and perform transactions.
- Anonymous Sites – The anonymous site service allows development of pages and forms to unauthenticated users. This is typically the part of an organizations website where customers do not need to identify themselves.

IV. PLATFORM ANALYSIS

In this study, we analyze several PaaS providers including Salesforce [17], Zoho CRM [18], SugarCRM [19], SuiteCRM [20], vTiger [21] and Heroku [22]. We chose the first five platforms because they each provide many of the services we discussed in detail. The last platform was added to show the difference between PaaS and PIaaS offerings. Each of the first five PaaS offerings was developed as a customer relationship management (CRM) system. The CRM vertical market software space requires integration with enterprise resource planning (ERP) systems. This integration requirement led the CRM vendors to develop their products as platforms instead of simply vertical market products. TABLE II shows the distributed services offered by each platform. Note the load balancing service is marked for the three PHP [23] platforms since the state of the session is stored in the database. Having the session state stored in the database allows additional business tier servers to be added to the configuration in the cloud. Though only Heroku has a graphical user interface (GUI) to manage node configuration, the PHP solutions can be hosted by an IaaS provider that provides the feature. TABLE II shows the application services offered by each platform. The final row shows a cost per user

TABLE II. DISTRIBUTED SERVICES BY PLATFORM

Service	Salesforce	Zoho	SugarCRM	SuiteCRM	vTiger	Heroku
Node Configuration						✓
Load Balancing			✓	✓	✓	
Logging	✓		✓	✓	✓	✓
Database	✓	✓	✓	✓	✓	✓
Scheduled Jobs	✓	✓	✓	✓	✓	

if the PaaS provider is providing both the infrastructure and application services.

V. Effort Study

To calculate the effort savings provided by the different PaaS service providers, we calculated the source lines of code (SLOC) in a stable platform release. For this study, we choose to use the SuiteCRM [20] system as our model. SuiteCRM is open source software, so we had access to the source code developed to provide the platform.

SuiteCRM is written in the PHP programming language using a MySQL database as its persistence layer. Using the debugger extension xDebug [24], we are able to trace all lines of code executed on the server when interacting with the application. xDebug plugs into the server-side execution of the code and creates a trace file with these lines of code. We developed tooling to parse the trace file and store the data in a MySQL database based on the function executed.

Due to the nature of Web application architectures, a single round trip from the Web browser to the Web server will often execute two distinct sets of functionalities. For example, when a user enters their login credentials, the POST to the server authenticates the user and then executes the code to display the homepage of dashboards. Our tooling allows a trace to add to or subtract from the functional cost. In the earlier example, we trace the combined functionality and then subtract the individual functionality of building the home screen.

For the study, we wanted the cost for local application software engineers in the Charleston, SC area. We recognize the cost for software developers changes from region to region but have an applied local example helps us present the work. The Bureau of Labor Statics [25] estimated the average cost for an application software engineer is \$96,200/year. Hadzima [26] estimates the cost of an employee's benefits and taxes at between 25% and 40% of base salary. On top of the salary cost, the employer must pay for rent for an office space, equipment, recruitment, training, etc. For our study, we are estimating the hourly cost of \$71.50 for an application programmer's time. In TABLE III, we show the estimated cost to pay an application programmer in the Charleston, SC area to redevelop the functionality provided by the service. For the study, we choose a specific platform that provided us with the source code so we could put a developer cost to the different services provided by PAAS providers. We analyzed SuiteCRM and looked at the source lines of code (SLOC). SuiteCRM stores the service source code in module folders on the file systems. We counted the executable lines of code and compared to the executed lines of code from the trace. Each trace represented a slightly higher number of lines of code because of shared libraries. We felt it was not appropriate to count all the shared lines of code per service, but we also felt it was not appropriate to ignore them completely. We decided to take the average between the two-line counts. We plugged the average number into the Constructive Cost Model (COCOMO) II formula [27] with our local application programmer cost of \$71.50 per hour. The fifth column in TABLE III shows the cost per service and the total cost of all services. We eliminated a few services from the study as the source code was not available. Note the table does not show the cost of the infrastructure services. The infrastructure services can be provided by an IaaS provider if the development is done to leverage the services.

TABLE III. COST PER SERVICES

Service	SLOC	Trace	Average	CoCoMoII
Authentication/ Authorization	1156	1437	1297	\$ 590,131
Workflow	954	1146	1050	\$ 467,789
Bulk Email	702	1054	878	\$ 384,246
Activity tracking	1178	1302	1240	\$ 561,674
Object Audit	656	873	765	\$ 330,225
Importing	1654	1857	1756	\$ 823,478
Exporting	945	1246	1096	\$ 490,375
Object Customization	2164	2874	2519	\$ 1,224,557
New Object Design	1474	1826	1650	\$ 768,981
Detail View	291	464	378	\$ 152,095
Edit View	1828	464	1146	\$ 515,031
Data Updates	656	989	823	\$ 357,860
User Interface Customization	2073	2482	2278	\$ 1,096,353
Multi-Select Fields	402	512	457	\$ 187,395
Report Display	1912	2356	2134	\$ 1,020,384
Report Creation	2957	3345	3151	\$ 1,566,362
Dashboard Display	1342	1672	1507	\$ 696,016
Dashboard Creation	1874	2198	2036	\$ 968,972
Web-services	986	1822	1404	\$ 643,884
Total				\$ 12,845,808

VI. Motivating Example

Imagine a software company wants to offer a new software solution to medium-sized entertainment venues such as local theaters, museum or minor league sporting attractions. Typically, the development stakeholders hold a tremendous amount of knowledge about the application domain they want to develop a solution for, but may not have either the knowledge or resources to develop the entire application architecture to deploy their application to the cloud. In fact, we would argue that they should hire programmers that can focus development on the domain-specific problems. In this example, the domain-specific problems include selling tickets from a limited inventory. The inventory may be assigned seats, general admission seats or some combination. There is also often a great deal of Management Information Systems (MIS) reports that are standard for the industry. These reports must be specified, developed and tested by the development team along with the software functionality for the data entry functions.

By leveraging a platform, the developers are able to spend their programming energy focused on the domain specific logic instead of application code that is standard across all business applications. This will reduce the costs, risks and the time to market.

VII. Platform Security

When a software application is built a set of functional requirements and non-functional requirements are normally created to help the developers to understand the expectations of the software. The functional requirements specify what the software must, and the non-functional requirements specify what must be true for the lifecycle of the software and the data created by the application. Most of our study to this point has focused on the functional requirements. Non-functional requirements tend to fall into three main categories:

- Performance and Concurrency – Non-functional requirements in this category specify how quickly the software must respond or how many simultaneous users the system must be able to support. A great example of this type of non-functional requirement is seen when looking at the construction of the healthcare.gov website in the United States [28]. The healthcare.gov website was developed to allow citizens of the United States to purchase individual health care through an online exchange. The system had a non-functional requirement of fifty thousand concurrent patrons. During the first week of the launch, the system was unusable because the non-functional requirement was not sufficient for the usage of two hundred and fifty thousand users.
- Data Correctness Constraints – Non-functional requirements in the data correctness constraint category include all the traditional relational database constraints including unique tuple identifiers, foreign key relationships and attribute domain values. A good example of this type of constraint from our motivating example above would be that every discounted ticket must be linked to a valid customer.
- Security – Non-functional security requirements tend to be less specific. We have briefly touched on authentication and authorization in our work above. Beyond those categories, almost every software application has a non-stated non-functional requirement that the software should not allow a malicious user to use the software in an unintended fashion.

Some of the service offerings compared in TABLE IV help with the first category of non-functional requirements. Specifically, the load balancing service is designed to allow the implementers to add additional server nodes to scale up the application to support more concurrent users.

For the broad category of non-functional requirement, we called security; we want to think about several different vulnerabilities and how a platform helps us to minimize or eliminate that vulnerability. We will think about the vulnerabilities in three categories:

- Injection Vulnerabilities – Injection vulnerabilities include attacks where either the malicious user can inject code into an application's page, or the malicious user is able to inject additional or altered database commands into a current database query.
- Forgery vulnerabilities – Forgery vulnerabilities allow an application to use the credentials the current operating user to gain access to an external resource.
- Redirection Vulnerabilities – Redirection vulnerabilities allow a malicious user to modify the URL that an application goes to after an action. This modification will often allow the malicious user to leverage the other two types of vulnerabilities.

A. *Platform Injection Attacks*

There are two types of injection attacks that a platform should protect an application from Cross-Site Scripting (XSS) and SQL injection. Cloud-based applications dynamically generate the user interface code (HTML, JavaScript, and CSS) that is rendered in a web browser. This is done through the cloud platform’s server-side programming language. The application will read data from the data store; parameters are passed in the request for the page and session state information is used to decide what code is placed into the user interface.

XSS is an injection vulnerability that exists when a malicious user can insert unauthorized JavaScript, HTML or other active content into an application’s user interface page. When an operator views the page, the malicious code executes and affects or attacks the operator. For example, a malicious script can hijack the operator’s session, submit unauthorized transactions as the operator, steal confidential information or simply deface the user interface of the application.

XSS attacks occur when operator input is reflected back in the user interface of an application page. This vulnerability stems from the poor separation between the code context and the user data. This poor separation allows the user input to be executed as code. There are three different types of XSS attacks that vary in the method in which the malicious payload is injected into the application and subsequently processed by the application.

- Stored XSS – This type of XSS attack occurs when the malicious input is permanently stored in the clouds data store and the code is reflected back to the user in a vulnerable application user interface page. A simple example where this type of attack often occurs in an application that displays a directory listing that shows users on the system. Any data stored in a user profile is stored in the cloud database and reflected back in the user interface listing.

- Reflected XSS – This type of XSS attack occurs when the malicious input is sent to a server and reflected back to the user on the response page. With this type of attack, the malicious user needs to convince the operator to click a hyperlink that has the malicious input connected to the link as a parameter. An example of a reflected XSS attack would be a hyperlink with JavaScript code in the parameter that attempts a Cross-Site Request Forgery (CSRF) to the hosted application.

- DOM-based XSS – This type of XSS attack occurs when a malicious payload is executed as a result of modifying the web page's document object model (DOM) in the operator's browser. The original application page is not modified, but the client-side code executes in a different way because of the changes in the DOM. In this case, the attack is done client side completely and not in the cloud. Many security techniques cannot detect this type of attack if the malicious input doesn't reach the cloud.

The second category of injection attack the platform should assist the developer in defending against is SQL injection. With SQL injection, a dynamically built database query is modified based on the input parameters of the operator to either expose private data or modify current data. Often, SQL injection attacks that modify data are a combination of both injection attack categories. The data being modified includes XSS code that will be rendered back to future users of the application.

B. *Platform Injection Defence*

When the server-side programming code merges either data from the database, a request parameter or a session variable, it needs to escape the variable, so any malicious code is no longer executable. There are three different types of user interface code that are vulnerable to injection attacks:

- HTML – The HTML represents the structure of the user interface page. This is the main area where merge fields are inserted by the server-side code.

TABLE IV. XSS ENCODING DEFENCES BY PLATFORM

Encoding Defense	Salesforce	Zoho	SugarCRM	SuiteCRM	vTiger	Heroku
Automatic HTML	✓	✓				
Disable Automatic	✓					
Programmable HTML	✓		✓	✓	✓	
Programmable JavaScript	✓					

- CSS – The CSS represents the style of the user interface page. It is infrequent that merge fields are inserted into the CSS, but it is possible and often an overlooked vulnerability.
- JavaScript – The JavaScript represent the client-side executable code. Again, it is infrequent that merge fields are inserted into the JavaScript, but it is also possible and also an overlooked vulnerability.

The defense for the XSS vulnerabilities in all three contexts is to encode the merge field so that malicious code is not executed. This follows the standard security mantra of "trust no-one." Do not trust parameters from the operator, do not trust data from the data store and do not trust variables from the session state. TABLE IV shows a comparison of the XSS encoding defenses provided by the platforms analyzed in our study. The first row of the table is for platforms that will allow all HTML to be encoded automatically. This is probably the best solution as it is not left to the programmer to remember not to trust the users. Unfortunately, sometimes the programmer needs to deal with the data directly, and automatic encoding causes issues. The second row from TABLE IV shows the platforms that allow the automatic encoding to be turned off on individual forms. The last two rows of TABLE IV show the programmatic functionality provided by the platform to the programmer. Programmable HTML encoding is a function in the library the developer must call and use the sanitized results returned from the function. Programmable JavaScript encoding is a function in the library the developer must call whenever a merge field is assigned to a JavaScript variable. This ensures that the malicious user did not inject additional JavaScript after the variable value.

The vulnerability with SQL injection stems from the developer programmatically building up of a SQL command by concatenating strings. The merge fields are typical values inserted between single quotes in the string. The platform defenses for the SQL injection attempt to sanitize the values appended to the string command. TABLE V shows the SQL injection defenses provided by the platforms in our study. The first defense in the study allows the programmer to encode the single quotes so that the merge field cannot turn a single value into a second command. The second defense in

TABLE V. SQL INJECTION DEFENCES BY PLATFORM

Defense	Salesforce	Zoho	SugarCRM	SuiteCRM	vTiger	Heroku
Single Quote Encoding	✓		✓	✓	✓	
Bind Variables	✓	✓				
No Command Execution	✓	✓				✓
No Modifiable Statements	✓					

TABLE V is a method that allows the merge field to be bound to a specific data type. This second method provides greater flexibility and protection. The last two defenses included in TABLE V stem from the legacy of the data store. If the data store is a traditional relational database in the cloud, there are vulnerabilities carried forward to the cloud. New solutions for cloud applications can protect against these vulnerabilities. Traditional relational databases allowed command execution on the server through the database engine. At implementation time, all databases could have this feature turned off, but the "No Command Execution" defense means the cloud provider removed this vulnerability. The final defense in TABLE V is if the cloud provider only provides an object-relational management (ORM) interface for data manipulation. This protects the systems by not allowing SQL injection attacks to modify the date.

C. Platform Redirection Vulnerabilities and Defence

Applications built in the cloud often follow a model-view-controller (MVC) design pattern. MVC allows your application to separate the user interface, the data that is persisted in the data store and the flow control of the application. We have already discussed security vulnerabilities in the user interface and the data store. With cloud architectures, an application's controller will often base the navigation in the application on a merge field from the user's request, session state or a database value. As seen with the user interface and database vulnerabilities, this can lead to problems.

In the best-case scenario, a malicious user can use the open redirect to open a page with a movie or cartoon on it. In the worst case, the malicious user can open a page that has an XSS or CSRF attack on it. There are three major defenses for open redirect attacks:

- Hardcode the URL – This defense limits the flexibility of the application by insisting that the programmer hardcode every URL in the controller.
- Local Only – This defense limits the flexibility by only allowing redirects to the same host and application.
- Whitelist – This defense requires the organization to build a list of acceptable URLs to redirect to.

TABLE VI. REDIRECT DEFENCES BY PLATFORM

Defense	Salesforce	Zoho	SugarCRM	SuiteCRM	vTiger	Heroku
Hardcoded flow		✓				
Local Only						
Whitelist						

Unfortunately, TABLE V shows the limited platform support for these three defenses. Only the Zoho CRM has any support, and it is because there is no programmatic access on the server side. With the Zoho CRM, custom objects follow the same flow for inserts, saves, deletes and cancellation. This lack of support leaves the programmers to implement the architecture strategy chosen for the product. Platform providers could make this easier by implementing a platform URL whitelist per application or by limiting the location for the redirects to local URLs only.

D. Platform Forgery Vulnerabilities and Defence

Modern web browsers allow the end user to have many tabs open in a single web-browser with cookies shared among browser tabs. This leads to a vulnerability called CSRF. As an example, imagine we have logged into a bank portal in one tab of my web browser. Typically, a website will write a cookie to identify the session on the server so future requests from my web-browser will not need to authenticate myself. If a malicious piece of code is running in another tab on the same browser, it can send a request to the bank website using the same cookie and gain access to my financial data. This example tries to make it clear that malicious code can gain access to data that should not be available to the malicious user, and CSRF allows the malicious user access to do just this. There are many less nefarious examples where the CSRF is attacking the same application and gaining access to or modifying data, not on the current user interface. Figure 1 shows a sequence diagram where malicious code gains access to an application called yourapp.com. In the scenario, a valid operator named Fred opens a web-browser tab to visit a

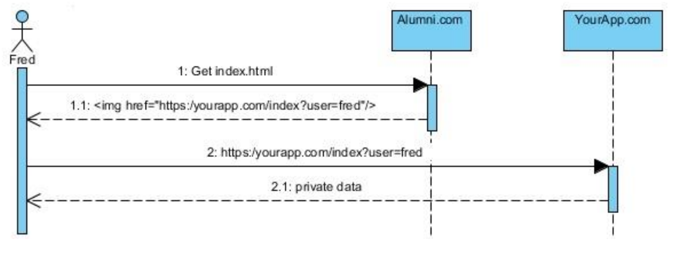
Fred
Alumni.com
YourApp.com
1: Get index.html
1.1: <img href="https:/yourapp.com/index?user=fred"/>
2: https:/yourapp.com/index?user=fred
2.1: private data

Figure 1.CSRF Attack

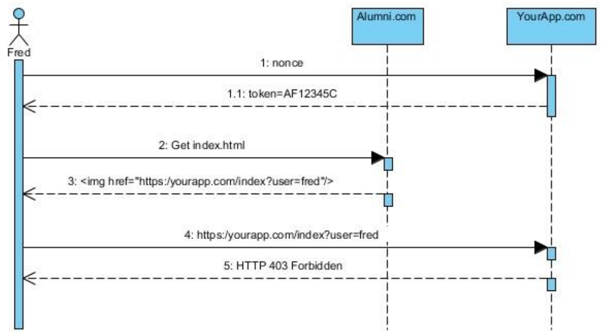

Figure 2.CSRF Defense

website named alumni.com. Alumni.com has an XSS injection where an image has a CSRF attack in the link for the image. When Fred clicks on the image, the malicious code uses his validated credentials on yourapp.com to gain access to private data.

There are two main defenses against CSRF, and both of the defenses require server-side validation of the request, allowing an even stronger defense.

- Server Side CSFR Tokens – In this defense the server generates a token per response. If the token is not returned with the follow-up request, then the request is considered a forgery and rejected.
- Refer Check – A standard header in the HTTP protocol is the REFER value. The refer value gives the URL of the page where the current request came from.

In Figure 2 we see a sequence diagram that modifies the earlier scenario to include a server-side CSFR token. The original request returns a token that needs to be included in the follow-up request. Typically, this is a hidden form field, so when the form has submitted the value of the tag is included in the HTTP post. If the HTML form code is returned by the server in one tab and the user toggles to another tab to visit another website, this tab will not have access to the one-time token.

TABLE VII shows the two types of CSRF defenses as implemented by the different platforms. As we saw with the open re-direct defenses, much more work can be done by the PaaS providers to make it easier for the application developers to focus on the business problem they are trying to solve. None of the providers apply both protections which would increase the protections for the applications with little work on the developer.

E. *Platform Clickjacking Vulnerabilities and Defence*

Clickjacking is an application vulnerability used by attackers to fool valid users into thinking that they are interacting with one object while they are actually interacting with a different object altogether. In a clickjacking-injected user interface, the malicious user shows content to the user while another form is on top of the content with a transparent layer. When interacting with the clickjacking user interface, the operators think they are clicking buttons corresponding to the bottom layer, while in reality, they are interacting with buttons on the hidden form on top. There are two common attacks that utilize clickjacking:

- Cursorjacking – This attack tricks operators into enabling the webcam and microphone on their machine through the Flash runtime engine.
- Lifejacking – This attack involves sharing or liking links on Facebook.

A common defense used to prevent clickjacking is called frame-busting. Frame-busting code is included on every page that stops a malicious user from loading your application in an iFrame. If the code detects the page is loaded in a frame, it will prevent the page from loading.

Another clickjacking defense is to use a relatively new HTTP header called X-FRAME-OPTIONS. This header is supported in all the latest web-browser version. The header defends the page in a similar way to the frame-busting code. The X-FRAME-OPTIONS header can be set to one of three values:

- DENY – This value does not allow the page from loading in a frame.
- SAMEORIGIN – This value allows the page to load in a frame only if the origin is the same as the content.
- ALLOW-FROM – This value allows the page to load in a frame only from a specific URL.

TABLE VIII shows the clickjacking defenses by each of the PaaS providers in our study. As we have seen with the past few attack defenses, the service providers have a long way to go to defend the platforms properly.

TABLE VII. CSRF DEFENCES BY PLATFORM

Defense	Salesforce	Zoho	SugarCRM	SuiteCRM	vTiger	Heroku
Server Side CSFR Tokens	✓	✓	✓			
Refer Check						

TABLE VIII. CLICKJACKING DEFENCES BY PLATFORM

Defense	Salesforce	Zoho	SugarCRM	SuiteCRM	vTiger	Heroku
Frame-busting	✓					
X-FRAME-OPTION	✓					

VIII. CONCLUSION

In this paper, we analyzed the programming effort required to reproduce services provided by a cloud PaaS provider. Our solution utilizes two methods to estimate the number of lines of code required for a service: SLOC and an execution trace. We utilize an average of the two methods to apply the COCOMO II costing algorithm. Our study demonstrates removing the need to develop the application services provided by the PaaS providers leads to a cost savings of nearly thirteen million dollars.

We also looked at non-functional services provided by the platform. In some cases, the platforms provided significant non-functional services; but in other cases, the platforms could do a much better job of providing the necessary protection.

In this research, we focused on application services provided by a PaaS, and future work needs to study the infrastructure services costs and the application development knowledge required to leverage the provided distribution services.

IX. ACKNOWLEDGEMENTS

We would also like to thank Kaitlyn Fulford who assisted on the earlier work this research has built upon.

REFERENCES

[1] A. Olmsted, "Platform As A Service Effort Reduction," in *CLOUD COMPUTING 2017 : The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization*, Athens, Greece, 2017.

[2] Google, "About Google Docs," 2017. [Online]. Available: https://www.google.com/docs/about/. [Accessed 10 Feb. 2017].

[3] Microsoft, "Office products," 2017. [Online]. Available: https://products.office.com/en-us/products. [Accessed 10 Feb. 2017].

[4] Amazon Web Services, Inc, "Amazon Elastic Compute Cloud - Virtual Server Hosting," 2017. [Online]. Available: https://aws.amazon.com/ec2/. [Accessed 10 Feb. 2017].

[5] Rackspace, "Rackspace.com - Rackspace® Managed Cloud," 2017. [Online]. Available: https://www.rackspace.com/. [Accessed 10 Feb. 2017].

[6] P. Mell and P. Grance, "The NIST Definition of Cloud," 09 2011. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. [Accessed 07 Sep. 2016].

[7] S. Kolb and G. Wirtz, "Portability in Platform as a Service," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, Oxford, United Kingdom, 2014.

[8] N. Baliyan and S. Kumar, "Towards Software Engineering Paradigm for," in *2014 Seventh International Conference on Contemporary Computing*, Noida, India, 2014.

[9] J. Ng, "Extending the Cloud From an App Development Platform into a Tasking Platform," in *2015 IEEE World Congress on Services*, New York, NY, 2015.

[10] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy and R. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," *Annals of Software Engineering*, vol. 1, no. 1, p. 57–94, 1995.

[11] M. Ray, "COCOMO II - Constructive Cost Model," 2016. [Online]. Available: http://csse.usc.edu/tools/COCOMOII.php. [Accessed 07 Sep. 2016].

[12] Oracle, "Oracle Database," 2017. [Online]. Available: https://www.oracle.com/database/index.html. [Accessed 10 Feb. 2017].

[13] Oracle, "MySQL Database," 2017. [Online]. Available: https://www.mysql.com/. [Accessed 10 Feb. 2017].

[14] Wikimedia Foundation, Inc, "NoSQL," 2017. [Online]. Available: https://en.wikipedia.org/wiki/NoSQL. [Accessed 10 Feb. 2017].

[15] Wikimedia Foundation, Inc, "Email spam," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Email_spam. [Accessed 10 Feb. 2017].

[16] A. Olmsted and G. Santhanakrishnan, "Cloud Data Denormalization of Anonymous Transactions," in *Cloud Computing*, Rome, Italy, 2016.

[17] Salesforce.com, inc, "Run your business better with Force.," 2006. [Online]. Available: http://www.salesforce.com/platform/products/force/?d=70130000000f27V&internal=true. [Accessed 03 Feb. 2016].

[18] Zoho Corporation Pvt. Ltd, "Zoho CRM is ready," 2016. [Online]. Available: https://www.zoho.com/crm. [Accessed 07 Sep. 2016].

[19] SugarCRM, "Discover a different kind of CRM," 2016. [Online]. Available: http://www.sugarcrm.com/. [Accessed 07 Sep. 2016].

[20] SalesAgility, "SuiteCRM – CRM for the world," 2016. [Online]. Available: https://suitecrm.com/. [Accessed 07 Sep. 2016].

[21] vTiger, "Grow sales, improve marketing ROI, and deliver great customer service," 2016. [Online]. Available: https://www.vtiger.com/. [Accessed 07 Sep. 2016].

[22] Salesforce, "Cloud Application Platform," 2016. [Online]. Available: https://www.heroku.com/. [Accessed 07 Feb. 2016].

[23] The PHP Group, "About PHP," 2017. [Online]. Available: http://php.net/. [Accessed 10 Feb. 2017].

[24] D. Rethans, "Xdebug - Debugger and Profiler Tool for PHP," 2016. [Online]. Available: www.xdebug.org. [Accessed 07 Sep. 2016].

[25] Bureau of Labor Statisics, "Occupational Employment Statistics," 2016. [Online]. Available: http://www.bls.gov/oes/current/oes_16700.htm. [Accessed 07 Sep. 2016].

[26] J. Hadzima, "How Much Does An Employee Cost?," [Online]. Available: http://web.mit.edu/e-club/hadzima/how-much-does-an-employee-cost.html. [Accessed 07 Sep. 2016].

[27] R. Madachy, "COCOMO II - Constructive Cost Model," [Online]. Available: http://csse.usc.edu/tools/COCOMOII.php. [Accessed 10 Feb. 2017].

[28] T. Mullaney, "Obama adviser: Demand overwhelmed HealthCare.gov," The USA Today, 5 Oct 2013. [Online]. Available: https://www.usatoday.com/story/news/nation/2013/10/05/health-care-website-repairs/2927597/. [Accessed 10 May 2017].