Understanding usage patterns of your ABL Application: Business and Technical Benefit

Coenie Vermaak

Solutions Architect – Britehouse Automotive

22 November 2017
It is impossible to develop the perfect application, that is, an application that is completely bug free, exceeds the end-users performance expectations, and contains all the required features.

However, it is more realistic to develop an application that is perceived to be perfect by focussing on the areas of the system that is used most by the end-users.
Why would we want to understand the usage patterns of our applications?

If you decide to do one of the following, it will be in your best interest to be aware of usage patterns:

• Modernization project
  • Better understand the “actual” or “real” scope of your project
  • Identifies a “training ground” for your project

• Architecture and / or UI design changes
  • Quantifies the impact of the change
  • Quantifies the load that can be expected

• Want to optimize or better understand your maintenance costs
  • Highlights the areas where you should spend your development and testing efforts
  • Identifies dead code that should be removed

It is important that you have a clear goal defined for the reason for tracking usage statistics
Is this “Usage Statistics Tracking” for me?

You might be thinking that you’ve got a stable system, running for years, and that there is no need to rock the boat

- You might be right, sometimes there is no need to understand and track system usage

For those of you saying, No there is absolutely no need for this usage statistics nonsense, Think about the benefit of knowing the following:

- The **frequency** a particular screen, function or peace of code executes
- What will the **impact** be, if we make a fundamental change in the system?
- Who is **actually using** the system or feature?
- How much of the system is **never used**?
- Which components are never used?

For those that is still not convinced – think of your software or application as an **asset**, can you with a fair degree of certainty quantify the **worth** of your asset?

- What about dead code and the impact of technical debt?
- Is your asset really worth what you think it is?
- Are you making a sound investment in your application by doing development and testing without understanding the real scope of the application?
How would you answer the following questions?

• Where should you focus your development and testing effort?
• Where do you start your modernization project?
Option 1:
Calculate the total quantum of your current application and deduce relative estimates
Ways to quantify your application

Traditionally the following methods are used to quantify an application:

• Counting lines of code
  • Can only be applied once the software or component is completed
  • Potential problems arise when considering how to deal with blank lines of code, comment lines, multiple statements per line and reused lines of code
  • Has a tendency to measure the size of the solution rather than the size of the problem

• Considering the total number of source files
  • Can only be applied once the software or component is completed
  • There is no indication as to the size and / or complexity of the contents
  • Volatile measure dependent on software architecture and heavily influenced by technical debt

• Counting Function Points
  • Functional size measurement considering the logical size rather than the physical size
  • Rather difficult and time consuming to calculate once the software or component is completed
  • Measurable during the design phase, however, it does not consider the technical complexity of the implementation

• Considering the total number of screens and / or interfaces
  • Functional measure hiding the complexity of the implementation
  • Can only be applied once the software or component is completed

• Considering the total size (on disk) of your application
Which is the best method...

- That really depends on who is asking and what you want to achieve

<table>
<thead>
<tr>
<th>Technical Bias</th>
<th>Business Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counting Lines of Code</td>
<td>Counting Function Points</td>
</tr>
<tr>
<td>Counting total number of source files</td>
<td>Counting Screens / interfaces</td>
</tr>
<tr>
<td>Total Size (on disk) of your application</td>
<td></td>
</tr>
</tbody>
</table>

- When considering hardware and technical requirements it would be more relevant to consider the *Total Size* of your application than it would be knowing either the total number of lines of code or the total number of function points in your application

- If however, you are tracking progress of a modernization project where a legacy system is refactored to a different architecture, technology or methodology, then it would make more sense to use the number of screens / interfaces, or the number of source files metrics rather than the number of lines of code or the total size of your application.

- Since the question at hand concerns development and testing effort as a whole, both Technical and business aspects are relevant
Consider the following Example

- Except for the *Total Number of Function Points*, the previously discussed metrics are fairly simple to calculate.

- The data can also be broken down per module or function and graphically represented to give us insight into the relative size of modules compared to each other.

- From an empirical point of view (and perhaps for some creative statistical modelling) the above information is interesting, but is this really useful?


<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total lines of code</td>
<td>2,813,867</td>
</tr>
<tr>
<td>Total number of source files</td>
<td>8,930</td>
</tr>
<tr>
<td>Total size on disk (MB)</td>
<td>450</td>
</tr>
<tr>
<td>Total number of screens</td>
<td>485</td>
</tr>
<tr>
<td>Total number of function points</td>
<td>?</td>
</tr>
<tr>
<td>Program</td>
<td>Count</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>Service</td>
<td>2755</td>
</tr>
<tr>
<td>Parts</td>
<td>2000</td>
</tr>
<tr>
<td>Vehicles</td>
<td>2000</td>
</tr>
<tr>
<td>Customers</td>
<td>912</td>
</tr>
<tr>
<td>General Ledger</td>
<td>800</td>
</tr>
<tr>
<td>CRM</td>
<td>200</td>
</tr>
<tr>
<td>Accounts Payable</td>
<td>128</td>
</tr>
<tr>
<td>Accounts Receivable</td>
<td>135</td>
</tr>
<tr>
<td>TOTAL</td>
<td>8930</td>
</tr>
</tbody>
</table>

Application Metrics - Number of Source Files

- What should be an evident flaw with this approach is the fact that we are considering none of the following:
  - Change Requests
  - Business Impact

- Even though probability theory indicates that since this is the amount of effort expended to date, it is likely that the same amount of effort will be required in the future, no realistic bias or seed is provided to the calculation and therefore the only real value we get from this metric, is understanding the size of the elephant, unfortunately no insight is provided for how we should eat the elephant

- It can be argued that some level of complexity is indicated by this metric – The greater the relative size, the more likely that:
  - Interrelated components exist
  - Higher levels of technical debt exist – which could indicate that a greater amount of code refactoring is required
  - A higher level of maturity exist in the module and therefore a greater chance that complex monolithic code exist

If you had a view of the usage metrics of the components, you would be able to quantify the Business Impact, as well as provide a better approximation of the relative size of the application
Option 2:
Evaluate the outstanding change requests and deduce relative estimates
<table>
<thead>
<tr>
<th>Enhancement Request</th>
<th>Performance Issues</th>
<th>Bugs</th>
<th>Effort Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>5</td>
<td>30</td>
<td>45</td>
</tr>
<tr>
<td>Parts</td>
<td>15</td>
<td>62</td>
<td>23</td>
</tr>
<tr>
<td>Vehicles</td>
<td>23</td>
<td>18</td>
<td>35</td>
</tr>
<tr>
<td>Customers</td>
<td>2</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>General Ledger</td>
<td>5</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>CRM</td>
<td>20</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Accounts Payable</td>
<td>4</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Accounts Receivable</td>
<td>3</td>
<td>17</td>
<td>6</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>77</strong></td>
<td><strong>196</strong></td>
<td><strong>127</strong></td>
</tr>
</tbody>
</table>

This approach provides the benefit of indicating potential work required relative to the total system whereas the previous approach indicated relative work completed compared to the total system.

In this instance there is a level of customer / end-user input into the potential development and testing effort.

The important metric that both the previous options overlooked or disregarded is the actual impact of the development effort on your customers’ business.

- This in turn could influence the risk profile for both you as the software vendor, as well as that of your customers.

In this case, having a view of the usage statistics will enable you to prioritize effort, as well as understand the impact of either investing or not investing the time and effort.
Let’s Recap

Where should you focus your development and testing effort?

<table>
<thead>
<tr>
<th>Option 1 - Total Quantum of System (Relative Size)</th>
<th>Option 2 - outstanding change requests (Relative Effort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability Theory indicates that the greater the relative size, the greater the development and testing effort will be in that area.</td>
<td>Scheduling Theory indicates that the optimal approach would be to start with the area with the least amount of effort and work your way through the issues.</td>
</tr>
<tr>
<td>The greater the relative size, the greater the complexity of the code and therefore more time will have to be spent refactoring and optimizing code, this in turn will lead to greater time spent on testing in this area.</td>
<td>From a user satisfaction- and data integrity perspective it would be desirable to address areas with the highest number of bugs and performance issues. However, from a commercial perspective a trade-off with Enhancement requests should be considered.</td>
</tr>
<tr>
<td>There is no measure of relevance or impact linked to your customers’ and therefore not the best option to base your development roadmap on.</td>
<td>In this instance, there is at least a measure of relevance to your customers’ business that is considered, however the impact on their business is still a mystery. It is also possible to make decisions based on commercial considerations.</td>
</tr>
</tbody>
</table>

Depending on the amount of information available to make the decision on where to focus your development effort, both options could be used as a basis to construct a development roadmap, However:

- Option 2 is more desirable than Option 1 since there is a relevance factor added to the planning
- Neither Option considers the potential Risks that would need to be managed
Where do you start your modernization project?

<table>
<thead>
<tr>
<th>Option 1 - Total Quantum of System (Relative Size)</th>
<th>Option 2 - outstanding change requests (Relative Effort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depending on the scope of the modernization project, this option provides some valuable insight into the complexity and potential scope of work</td>
<td>Since modernizing software comes with its own challenges and complexities, this option can identify areas that are relatively stable or used infrequently, to use as a starting point</td>
</tr>
<tr>
<td>No insight into customer benefit or impact is indicated by this option</td>
<td>Although a small measure of customer input is used by this option, no real benefit or impact is indicated by this option</td>
</tr>
</tbody>
</table>

Considering the complexities of a modernization project it would be wise to combine the output of the two different options when answering this question:

- We know from probability theory that the greater the relative size, the greater the effort required
- We know from scheduling theory that the optimal schedule would be to start with the least amount of effort
- Combining the above theories with set theory would provide a comfortable happy medium, that is, by taking the union of the least amount of effort and the smallest relative size would be a really good starting point

**From a Risk mitigation point of view:**

Learning and platform implementations should be done on non-core or business critical areas of the application. Understanding the usage patterns of your application will assist in identifying suitable starting points to ensure that the following is put to bed before business critical components are touched:

- Architecture, Framework selection and implementation, Deployment strategies, etc
Nash Equilibrium

“A stable state of a system that involves several interacting participants, in which no participant can gain by a change of strategy, as long as all the other participants remain unchanged”

• If we lived in a world where we could reach a Nash Equilibrium with our competitors then the options provided previously would be sufficient, however in the software market today it is imperative to provide solutions to our customers that:
  • Is cost effective
  • Provides effective management and mitigation of risks
  • Relevant to their business
  • Provides a competitive advantage

• One of the most relevant pieces of information, that should form the basis of our decision making, is the usage patterns of our applications
Understanding Usage Patterns of your ABL Application: Production

Usage Statistics provides empirical data that highlights the following:

• How the software is used in the “Real World”
  • Helps identify potential deficiencies in business processes
  • Better understanding of usage patterns provides more insight to create a rational development roadmap

• How different customers use the software
  • When an understanding of your customers’ business is attained the following becomes possible:
    • Custom reporting requirements and potential Alerts requirements become evident
    • End-User training and documentation requirements can be better understood

• Which components of the system are not used and can be decommissioned

• Usage volumes of the components and / or modules
  • Better understand how to spread development competencies
  • Provides insight into the potential risks and benefits of software changes on your customers’ business
  • Enables you to quantify the impact of bugs and performance issues on your customers’ business
  • Assists in software design and optimizations of business processes and business logic

• Potential vulnerabilities related to user permissions and privileges
  • Audits are concerned with segregation of duties
Important Considerations

• Define the scope for analysis
• Start small and grow your usage tracking as required
• Implement the usage tracking in a way that will minimize additional overhead on the system
• Collect data in a way that makes it is easy to collate data from all customers or production sites
• Ensure that you derive value from all statistics - **Collect only data that will aid in decision making**
Real World Example
• **What did we want to achieve?**
  • The primary objective was to develop a roadmap for our modernization project
  • Understand the total production quantum of our application
  • Quantify the impact of our modernization project on our customers and ensure that we are managing this appropriately
  • Identify the high-impact areas of the application
  • Create an audit of user access to validate and monitor our permission scheme
  • Understand who is using the system, how often and why

• **What data did we collect?**
  • It was decided that collecting data when screens are accessed, would be sufficient to achieve our goals
  • Whenever a screen is opened we track the User Identifier, Screen Name, Module, Company Identifier and increment a Screen Count
  • As a second metric we analyzed the permissions that provide access to screens
  • Data is stored centrally, aware of customer segmentation (but not driven by), to provide an overview of the global usage patterns for the application.
### How did this help us?

<table>
<thead>
<tr>
<th>Screen Title</th>
<th>Total Usage %</th>
<th>AM Usage %</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Centre</td>
<td>12.83%</td>
<td>6.59%</td>
<td>✔️</td>
</tr>
<tr>
<td>Parts Invoicing</td>
<td>11.18%</td>
<td>3.08%</td>
<td>✔️</td>
</tr>
<tr>
<td>Bulk Oil Interface</td>
<td>7.78%</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>General Postings</td>
<td>7.37%</td>
<td>3.53%</td>
<td>✔️</td>
</tr>
<tr>
<td>Parts Inventory System</td>
<td>7.71%</td>
<td>2.85%</td>
<td>✔️</td>
</tr>
<tr>
<td>Available Printers</td>
<td>5.36%</td>
<td>0.61%</td>
<td>✔️</td>
</tr>
<tr>
<td>Parts Orders Browser</td>
<td>4.04%</td>
<td>1.88%</td>
<td>✔️</td>
</tr>
<tr>
<td>General Ledger System</td>
<td>3.99%</td>
<td>12.33%</td>
<td>![x]</td>
</tr>
<tr>
<td>Accounts Receivable System</td>
<td>3.89%</td>
<td>2.85%</td>
<td>✔️</td>
</tr>
<tr>
<td>Accounts Payable System</td>
<td>2.99%</td>
<td>2.35%</td>
<td>✔️</td>
</tr>
<tr>
<td>Process Receipts</td>
<td>2.87%</td>
<td>0.52%</td>
<td>✔️</td>
</tr>
<tr>
<td>Admin Orders</td>
<td>2.87%</td>
<td>1.74%</td>
<td>✔️</td>
</tr>
<tr>
<td>Parts Order Details</td>
<td>2.52%</td>
<td>0.48%</td>
<td>✔️</td>
</tr>
<tr>
<td>Vehicle Inventory System</td>
<td>2.45%</td>
<td>5.31%</td>
<td>![x]</td>
</tr>
<tr>
<td>Vehicle Deal Builder</td>
<td>2.10%</td>
<td>3.05%</td>
<td>![x]</td>
</tr>
<tr>
<td>Add Orders to Ship Doc</td>
<td>1.55%</td>
<td>0.31%</td>
<td>✔️</td>
</tr>
<tr>
<td>Parts Lookup</td>
<td>1.35%</td>
<td>0.14%</td>
<td>✔️</td>
</tr>
</tbody>
</table>

• Only 70% of the Total system is being used
• 80% of the system usage occurs in 5% of the system (17 Screens)
• 48% of our Customer Care department’s time is spent on the 5%

---

![Screen Title Ave Per Month Ave Per Day Ave Per Hour](image)
- Using the Usage statistics metrics we can determine the number of users per company / customer that use each screen

- Why is this important:
  - This information is vital in understanding the impact on, and training requirements for end-users when new functionality or a fundamental change in process is introduced
  - Understanding usage per company allows us to understand the impact on deployment and allocation of resources to assist end-users
What are some of the implications

Modernization Project

- 30% of the system, as we previously quantified it, is dead code, and therefore does not have to be refactored
  - At a minimum, that means that 147 screens and some of the associated backend logic will not have to be refactored
  - Making a rough estimate (disregarding complexity) and assuming that the average refactoring time of a screen is 5 days, this equates to 5 880 hours worth of work, and at an average of 30 Euros per hour, this is a direct development saving of 167 000 Euros
  - In “Frequently Forgotten Fundamental Facts about Software Engineering”, Robert L. Glass (article in IEEE Software May/June 2001), talks about software’s “60/60” rule, that is, maintenance typically consumes 40% to 80% (60% average) of software costs
    - Therefore we can realize a substantial development cost saving

- Focusing on 5% of the system (17 Major functions) will realize the benefits of modernization for 80% of the end-user interactions
  - Allows us to create a rational software roadmap based on empirical data
  - Resource allocation can be managed proportionately to the potential impact on the system, that is, senior developers are assigned to work on the “Hard-hitting” 5% and junior developers can be used for the less critical components

- The average usage statistics gives us insight into the areas of the system where performance must be a primary consideration when refactoring code
General Implications for Development and Development planning

• Quantifying the impact of bugs and performance issues on your customers’ business
  • Naturally, any bugs or performance issues related to the 80% of system usage will have a massive impact on your customers’ business and therefore should be prioritized for resolution
  • Conversely, issues related to the bottom quartile of system usage can be deferred and therefore a more realistic development pipeline, guaranteed to be in the best interest of your customers’, can be attained

• Usage Statistics provides empirical data to develop and maintain a realistic and effective risk management strategy
  • This includes managing the following risks:
    • An increase in development cost
    • The development of poor quality software
    • Mismanaging the natural attrition of a program or piece of code – over time maintenance and enhancements can lead to technical debt and therefore a degradation in quality and performance
    • Managing the necessity and implementation of customizations and enhancements

• Provides empirical data that allows answering questions like,
  • Is it possible or feasible to modernize the entire “view / application” for a group of users?
General Implications for software testing

Understanding the usage patterns of your application or system is vital for planning your testing strategy

- Due to the ever increasing requirement for shorter software delivery cycles, it has become crucial to understand where and in what proportion to spend your testing time
- It is very seldom the case that all components of the system or even all implicitly impacted areas of the system contained in a release is tested thoroughly
- Accepting this fact, it is more desirable to defer the risk to the areas of the system that is part of the least used components
- Therefore, understanding the usage patterns of your system, allows for the application of probability theory to assist in determining the components that must be tested, as well as the proportion of time to be allocated for testing the components
- Components suited for implementation of automated testing can be selected based on empirical information

Understanding the usage statistics of your system can be used to determine the required depth of testing, and not only the required width, that is, covering more permutations of scenarios rather than more screens or functions
How Usage statistics can influence your approach to testing

It is obvious to see the benefit of tracking usage statistics in your testing environment as it serves as an audit of your testing process.

<table>
<thead>
<tr>
<th>Screen Title</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Centre</td>
<td>12.81%</td>
</tr>
<tr>
<td>Parts Invoicing</td>
<td>11.27%</td>
</tr>
<tr>
<td>Bulk Oil Interface</td>
<td>7.95%</td>
</tr>
<tr>
<td>General Postings</td>
<td>7.36%</td>
</tr>
<tr>
<td>Parts Inventory System</td>
<td>5.90%</td>
</tr>
<tr>
<td>Available Printers</td>
<td>5.39%</td>
</tr>
<tr>
<td>Parts Orders Browser</td>
<td>4.07%</td>
</tr>
<tr>
<td>General Ledger System</td>
<td>4.02%</td>
</tr>
<tr>
<td>Accounts Receivable System</td>
<td>3.90%</td>
</tr>
<tr>
<td>Accounts Payable System</td>
<td>2.99%</td>
</tr>
<tr>
<td>Process Receipts</td>
<td>2.88%</td>
</tr>
<tr>
<td>Admin Orders</td>
<td>2.81%</td>
</tr>
<tr>
<td>Parts Order Details</td>
<td>2.55%</td>
</tr>
<tr>
<td>Vehicle Inventory System</td>
<td>2.47%</td>
</tr>
<tr>
<td>Vehicle Deal Builder</td>
<td>2.08%</td>
</tr>
<tr>
<td>Add Orders to Ship Doc</td>
<td>1.56%</td>
</tr>
<tr>
<td>Parts Lookup</td>
<td>1.36%</td>
</tr>
</tbody>
</table>

- Doing performance testing on every screen or function in your system is not realistic, and generally issues only become apparent in production.

- Considering the usage statistics of your system, allows you to have the best of both worlds, that is, you can focus on the high-impact areas, without adding major workload on your testing and development teams.

- The benefits of focusing your efforts includes:
  - It becomes easier to manage the responsiveness of your application for the majority of daily operating processes.
  - The impact of a major performance issue being released to production is minimized and therefore assists in managing risks associated with software deployments.
  - Overall customer satisfaction can be improved by ensuring that the components that is used most often, or by the most users, is responsive and by ensuring that no major issues is introduced in those areas, a perception of an “Excellent Application” can be maintained.
Some interesting facts we’ve learned

- The positive is that our testing efforts are at least in the ballpark
- Arguably a negative was that, some of the components we know to be major components, did not receive the focused attention that if could have – Our testing was scattered more evenly over a greater function set
- Analysis of our test coverage indicated that some areas of the system that has been used by the end-users was not tested for a substantial period of time
  - This was due to testing being driven by development changes and therefore the “unaffected” areas was not tested – an obvious problem with this approach is the fact that those areas could have been impacted implicitly and would have gone undetected
  - Perhaps the difference between tested- and used functionality is a good source for automated testing?
Implicit benefits of understanding the usage patterns of your application

- **Implications with regards to your customer base**
  - Improving productivity is one of the most significant ways to succeed in business, we need our customers to work smarter, faster and more competitively
  - You have empirical evidence of the core areas of functionality for your customers
  - Enables you to make rational decisions about:
    - Proactively enhancing system processes to derive value for your customers
    - Whether an enhancement request is genuinely bespoke or will be for the greater good of your customer base
  - Provides vital information to tailor your licensing model
    - Perhaps an additional license type can or should be introduced, at a reduced cost, that will enable you to sell more licenses

- **Implications with regards to training and documentation**
  - It is highly unlikely that your entire system will be used on a daily basis, therefore end-user training can be tailored to focus on specific levels at specific times
  - Initial implementation training can focus on core areas ensuring that end-users can use the system
  - Additional training can be provided at the required times for remaining functionality
  - Documentation can be adapted to perhaps a quick-how-to for core functionality where extensive training is provided and more focused tutorial type training documentation can be provided for non-mainstream functionality
• Implications for non-technical managers and executives

Making choices as a manager or executive involves understanding the trade-off between the cost of minimizing system defects and the financial benefit of doing it.

• It provides a view of how your application is actually being used
• It provides a means to recognize and understand the impact of performance issues on the system
• It provides a view on how to best address and prioritize performance issues and system defects
• Highlights the possible future consequences of not thinking ahead and optimizing the system now
• Understanding the actual staff turnaround of your customers, will provide a view on the necessity for the investment in:
  • UI Design and simplification of processes
  • Potential training requirements and opportunities
  • The time to tailor documentation and documentation platforms
• It provides a means to recognize and understand the impact of releasing software that is not thoroughly tested and production ready
(Revisit)
How would you answer the following questions?

- Where should you focus your development and testing effort?
- Where do you start your modernization project?
Focus of Development and Testing Effort

The total percentage of usage combined with the number of impacted users and the effort required are critical metrics that should be considered, as this has a direct correlation with customer satisfaction and risk management.

In the current example, probability theory combined with the impact determined by usage and users impacted, it is clear that development and testing focus should be placed on the parts module, followed by the service module, the General Ledger module, and so on and so forth...

<table>
<thead>
<tr>
<th>Module</th>
<th>Usage</th>
<th>Effort Required</th>
<th>Relative Size</th>
<th>Users Impacted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>12.83%</td>
<td>20%</td>
<td>30.85%</td>
<td>2466</td>
</tr>
<tr>
<td>Parts</td>
<td>34.13%</td>
<td>25%</td>
<td>22.40%</td>
<td>3198</td>
</tr>
<tr>
<td>General Ledger</td>
<td>17.10%</td>
<td>8%</td>
<td>8.96%</td>
<td>385</td>
</tr>
<tr>
<td>Accounts Payable</td>
<td>3.89%</td>
<td>5%</td>
<td>1.43%</td>
<td>168</td>
</tr>
<tr>
<td>Accounts Receivable</td>
<td>2.99%</td>
<td>7%</td>
<td>1.51%</td>
<td>182</td>
</tr>
<tr>
<td>Vehicles</td>
<td>4.55%</td>
<td>19%</td>
<td>22.40%</td>
<td>851</td>
</tr>
</tbody>
</table>

Where to start modernization project

In the case of modernization projects, where new technology and different user-interaction paradigms are typically introduced, it would be wise to start with the areas of least impact to end-users.

Once the foundation is laid, a rational decision can be made based on the desire to either complete as many components in as short a time as possible, or to complete as much of the system per iteration as possible.

Having the additional metrics at our disposal, allows us to make rational multi-dimensional decisions, based on complexity, customer satisfaction, resource availability, risk management and cost.
How can usage statistics be used as a disruptor in your industry vertical?

- Can you get a competitive advantage by changing your licensing model from a SaaS model to either a transaction-based or consumption-based model?
- Can you provide better TCO for your customer by cutting development and testing costs without compromising on quality?
- Can you beat your competitor to market with product enhancements and value-adds by focusing on “Actual” core areas rather than “Perceived” core areas?
Summary

• Understanding usage patterns provides a critical dimension for understanding the total quantum of your system or application

• Ultimately the software we develop are meant to assist our end-users in solving a particular problem in an efficient and cost effective manner

• Analyzing and understanding usage patterns provides a platform to effectively management and mitigate risks

• Provides an understanding of how your application is being used and which components of the system are being used

• More informed decisions regarding modernization projects, bespoke development and system enhancements can be made

• The impact of performance issues and system defects can be rationalized and dealt with in an informed priority

• Software testing can be tailored to maximize coverage of actual critical components whilst minimizing the effort of doing so

• Ensure that the metrics chosen to understand the usage patterns of your application is relevant and adds value
  • Small datasets can provide valuable information, but might not provide answers to all the questions
  • Grow usage analytics over time, avoid “big bang” implementations that could provide unnecessary overhead to potentially and already strained system
  • Ensure that it is possible to turn the tracking system on and off as required
Any Questions?

Illuminating what’s next™