Software: The Brains Behind U.S. Defense Systems

As the U.S. military shifts its focus from metal and mechanics to unmanned vehicles, drones, and smart bombs, software is becoming a crucial piece of weaponry.
Technology is a strategic weapon across virtually all industries and organizations. Nowhere is this more true than in the military and the U.S. Department of Defense (DoD). The DoD is well known for its hardware-based weaponry, unmistakable in aircraft, satellites, missiles, and other systems in its arsenal for defending the country. Now, the military is shifting its emphasis from hardware to software. More specifically, the DoD wants to integrate and better manage the technology and software that play an ever-increasing role in modern weapons systems and national security.

This shift will require the DoD to alter its perspective, processes, and capabilities to avoid the increasing costs associated with software development, modernization, and sustainment. Failure to do so will burden the country’s capabilities, forcing the military to operate under complex and tangled systems that will cost billions of dollars to revamp. As an Air Force general explains it, “The B-52 lived and died on the quality of its sheet metal. Today our aircraft will live or die on the quality of our software.”

As advanced software systems and embedded software technologies become the brains behind modern warfare, it is pushing military capabilities to develop and maintain these complex architectures. When measured by source lines of code (SLOC) created or modified by software developers, the amount of software code in modern war-fighting systems has increased significantly over the past decade (see figure 1).

Like it or not, the DoD is now in the software business.

**Figure 1**
The number of source lines of code (SLOC) has exploded in avionics software

<table>
<thead>
<tr>
<th>SLOC in thousands</th>
<th>Operational software</th>
<th>Operational and support software</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-16A Block 1 (1974)</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>F-16D Block 60 (1984)</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>F-22 Raptor (1997)</td>
<td>1,700</td>
<td></td>
</tr>
<tr>
<td>F-35 Lightning II (2006)</td>
<td>6,800</td>
<td></td>
</tr>
<tr>
<td>F-35 Lightning II (2012)</td>
<td>10,000</td>
<td></td>
</tr>
<tr>
<td>F-35 Lightning II (2012)</td>
<td>24,000</td>
<td></td>
</tr>
</tbody>
</table>

Notes: SLOC for F-16 and F-22 are at first operational flight. F-35 SLOC figures are from first test flight and current estimates/sources.
The Business of Software

The increased demand for software stems from an emphasis on advanced operational capabilities and requirements. For example, the airframe for advanced fighter jets has reached the limit of modern aerodynamic improvement. This limitation, coupled with recent advances in remotely piloted aircraft (RPA), places greater demands on operational flight programs (OFPs) to bring new capabilities to bear rapidly. Similarly, smart weapons—such as the GPS-guided Joint Direct Attack Munition (JDAM), laser-guided munitions, and AIM-9X—require continual software updates to stay smart. Full autopilot, launch accessibility region calculations, and other advanced capabilities can only be enabled using increasingly sophisticated algorithms and software architectures.

“The B-52 lived and died on the quality of its sheet metal. **Today our aircraft will live or die on the quality of our software.**”

Additionally, ground systems must overcome the challenges associated with deploying large-scale software solutions. The Army recently canceled the Future Combat Systems (FCS) program due to budget and schedule issues that were largely the result of complex software development efforts. A recent review of the F-35 Joint Strike Fighter technical baseline underscored the challenges of FCS and many other military programs, illustrating the difficulties program offices face in managing, adapting, and even comprehending software development initiatives.¹

With shrinking defense budgets and increasingly complex system requirements, government leaders and key contractors are struggling to deliver advanced software-enabled systems affordably. The shift from hardware to software is particularly challenging for modernizing systems and managing sustainment.

Defense System Software: Looking Ahead

The defense and software landscape is dynamic, with several technological, programmatic, and enterprise barriers, among other issues, having a major impact over the next decade (see figure 2 on page 4). Knowledge in the following areas will promote the affordable development, modernization, and sustainment of modern weapons software systems:

**The architecture advantage.** Determine the optimal architecture decisions and impact early in the design and planning processes, and continue assessments through the maintenance phase.

- Recognize that selecting the proper software architecture is a cost-effective way to rapidly improve capabilities
- Understand what impact modular, federated, and integrated architectures have on weapons systems and future capabilities
- Determine which architectures allow for cost-effective enhancements

The commercial software advantage. The DoD uses commercial software just like the private sector—many functions are best served with these products to improve capabilities, accelerate deployment, and improve total cost of ownership.

- Evaluate how and where commercial software can be deployed to use the latest capability at the lowest cost
- Synchronize, update, or utilize refresh cycles to take advantage of the latest commercial software functionality
- Use commercial software in both critical and non-critical systems as a replacement for costly custom development

The should-cost analysis advantage. Perform should-cost analyses to estimate costs more accurately for the software capabilities being produced, integrated, and tested—knowing this can save millions of dollars on key projects.

- Validate contractor proposals by independently estimating the cost of software development
- Identify improvement opportunities for software developers to reduce costs and deliver benefits to the program and taxpayers
- Ascertain what software models and comparisons are necessary to get the most accurate cost estimate
The affordable sustainment advantage. Outline a clear strategy for the development of organic capabilities needed to maintain the weapons systems—and do so affordably.

- Add new capabilities to existing weapons systems affordably to maximize asset value
- Determine the right partnership between industrial and organic sustainment efforts
- Build an organic capability to support next-generation weapons systems
- Optimize investment in software maintenance and modernization

The Architecture Advantage

How various software-enabled functions are designed and constructed within a weapons system determines the degree of effort required to enhance or modify software and deliver new capabilities. Some integrated architectures are so tightly coupled that making even small changes brings significant costs. However, if modularity and loosely independent design are applied from the start, maintenance and enhancement efforts will be more efficient.\(^2\) One approach to affordability promotes the use of federated avionics architectures, such as the one exhibited on recent fifth-generation fighter jets.

From remotely piloted aircraft and smart bombs to autonomous vehicles and advanced fighter jets, **software is crucial to the success of today’s weapons systems.**

Federated architecture. Federated architectures allow information to be shared among modules. The design is component-based and similar to that of the old legacy systems. Federated systems typically have a central processing function (CPF) that coordinates information across the platform. In avionics software, the CPFs do not assume responsibility for data processing but instead organize information necessary for pilots to make decisions, while each subsystem gathers and processes data from its relevant sensors. A federated architecture delivers the same advanced capabilities and sensor fusion of highly integrated architectures but with somewhat reduced overall processing efficiency because of its modularity. However, today’s modern hardware has ample capability and speed to overcome any reduced processing efficiency of federated architectures.

Despite decreased efficiency, modular components can be upgraded quickly and independently, thereby reducing maintenance costs. Because of the nature of designs featuring tight coupling, integrated architectures require significantly more effort to develop and maintain across the software development life cycle. A federated architecture sustains lower overall effort and associated costs of software designs by a factor of two or more when compared to an integrated architecture (see figure 3 on page 6).

\(^2\) Modularity is the degree to which a system is made up of independent units that can be combined into a larger application.
Open architecture. The next significant evolution in avionics architecture will rely on the principles of open architecture, which offer an alternative to today’s multitude of proprietary standards. Open architecture is a type of hardware or software design that allows adding, upgrading, and swapping components that conform to agreed-upon standards. Open architecture allows independent parties to design and develop interoperable components that work together under the specified standards. By applying similar functions or skills at the platform level, greater specialization and flexibility becomes possible among developers across multiple contractors—significantly reducing the cost and effort of any software project.

Open federated architecture. A federated architecture is the natural model for open architecture development. The combination of the two paradigms (open and federated) increases the ability of development teams—and even encourages them—to modularize their code and compartmentalize their efforts.

Already, major contractors are developing their own native open architecture standards and models focused on subsystem development. However, the government should promote the development of a general open architecture, supported by all involved contractors across
entire systems. Such a coordinated effort would encourage general adoption. The DoD should focus early design on these open architectures while striving to achieve needed performance capabilities for weapons systems. Great benefits can be realized by investing critical time early in the architecture design phase with contractor and government software professionals who will maintain the system and live in the future with today’s choices.

The Commercial Software Advantage

Just as private sector companies have moved to commercial software and enterprise resource planning (ERP) systems over the past two decades, the military is now undertaking a similar transition. When appropriate, using software developed for commercial use instead of expensive, time-consuming, custom-developed software can bring the latest capabilities to the battlefield and support systems at a fraction of the cost. Using commercial software in non-critical applications allows proven best practices to be adopted, thus avoiding problems already solved commercially.

Software developers are pushing the limits of avionics in modern systems to keep older aircraft competitive.

In recent years, commercial software has become more attractive for modifying or retrofitting existing platforms for future weapons systems. Commercial software applications and processes draw from a larger pool of experience to speed up development at a lower cost. Additionally, commercial platforms offer advanced capability and increased performance over custom or embedded systems, which often rely on older technology.

As an added benefit, the extensive use of commercial software presents an opportunity to coordinate upgrades across weapons platforms by sharing common architectures, code, and maintenance efforts. Commonality across systems will allow for more centralized training of both in-house maintainers and operators, increasing the flexibility of personnel across the services, which is critical in times of conflict or force reductions. Additionally, larger purchases of commercial software will offer significant negotiating power because of the supplier’s near-zero marginal cost of producing or delivering additional units.

Transitioning from custom to commercial software begins by understanding their basic differences. Commercial software has four key advantages:

**Faster development cycle.** Commercial software is developed much faster than traditional DoD software systems, with comparable system developers 50 to 100 percent more productive. The shorter cycle allows advanced capabilities to be fielded faster and development teams to respond swiftly to requests for changes.

**Advanced capabilities.** Modern software development methods are changing rapidly, providing developers with a wider range of tools for coding applications. This allows for the use

---

of the latest techniques to deliver applications that would be impossible to develop with legacy software systems.

**Lower cost.** Only a few software developers have proficiency with DoD custom development standards. As a result, it is expensive to retain a large number of highly experienced developers for even basic projects. Recruiting from the much larger pool of developers with commercial software experience will drive down costs to market rates and yield a more agile pool of developers.

**Continual refresh.** Commercial software undergoes continual maintenance, which leads to new capabilities with each software release. These upgrades are aligned with functional best practices and offer customers the benefits from global leaders that create leading and scalable functionality for end-users and organizations.

Capitalizing on the benefits of commercial software requires adopting a faster, more coordinated refresh strategy. A hardware refresh that is closer to the three- to five-year cycle for commercial platforms should replace the traditionally longer defense modernization schedules, which take 10 or more years for many DoD projects. A shorter cycle ensures that large investments in warfighting systems remain operational throughout their lifetime.

Also, while a shorter cycle requires better coordination between software and hardware releases, it also allows taking advantage of historical performance advancements and pricing improvements in commercial hardware and infrastructure (see figure 4).
Executing this more rapid refresh cycle will require secure sources of regular funding to reduce the total cost of ownership. If funding is not provided for a full upgrade and the modification cycle lags, there will be additional costs to maintain legacy configurations. Only a stringent schedule of regular updates will manage risk and reduce overall costs.

To summarize, the use and effective integration of commercial software is not only attractive but a reality for DoD and weapons systems. Improving commercial hardware and software solutions can greatly enhance capabilities, increase the speed to deployment, and significantly drive down costs. The DoD’s ability to recognize this shift and manage the vendors and solutions will be critical over the next decade and beyond.

The Should-Cost Analysis Advantage

For most software development projects, estimating the required cost and effort can be challenging. Procurement and acquisition organizations—long the experts at estimating hardware-related projects, upgrades, and modernization efforts—often struggle to estimate and manage software development costs.

An inaccurate estimate can cost time, money, and lost capabilities. It is difficult to estimate software development efforts largely because of the complexity of the domain coupled with inadequate sizing estimates, lack of experience with similar efforts, and poorly defined requirements. Poor software size estimation is a main reason many major DoD programs exceed costs and fail to meet deadlines.

Should-Cost Modeling

A should-cost analysis provides insight into cost drivers (for use in planning and negotiations) and forces accountability, especially across contracts and suppliers—shedding light on the overall development life cycle. A typical should-cost software estimation includes the following four elements:

- **Parametric modeling.** Models based on SEER for Software (SEER-SEM) and Constructive Cost Model (COCOMO), which rely on industry standards drawn from experience across multiple projects, help estimate the software effort and determine optimal schedules.

- **Contractor comparison.** Industry benchmarks are a good way to estimate what software should cost. Examining the performance of similar contractors helps develop an understanding of best practice performance.

- **Program comparison.** Benchmarking similar services and common mission profiles across a comparable scope of work can help estimate costs.

- **Bottom-up estimation.** Each activity is modeled based on a discrete view of component costs against specific industry-standard benchmarks and cost drivers.

  - **Technology (software packages),** broken down into the smallest components to gain insight into the system as a whole.
  - **Once the system has been defined at its lowest possible level—typically down to individual programmer teams or subsystem components—cost drivers are applied against specific measurements and activities. A typical measurement might be hours required to develop an equivalent source line of code (ESLOC) for a given team. The cost drivers model the effort required for a team to develop a particular piece of software. Summing these cost estimates over each team, phase, and software package provides a full view of the effort required across people, processes, and technology.
Case Study: U.S. Air Force Avionics Should-Cost Analysis

Aerospace engineers and designers have pushed the bounds of the possible with the mechanical design and construction of military aircraft, creating machines that are both breathtakingly beautiful and devastatingly effective. Now, software developers are pushing the limits of traditional avionics in modern systems to keep older aircraft competitive and create innovative capabilities in the new ones.

The DoD and the Air Force recently needed a streamlined process to evaluate how much an aircraft software modernization effort should cost. Our work with the DoD and the U.S. Air Force resulted in sizable savings for all parties, including taxpayers.

Approach
The analysis involved the processes and costs of a proposed half-billion-dollar modernization effort for the Air Force fleet. This software-only modernization effort involved adding several new tactical capabilities to ensure the success of current and future missions.

The project began with an extensive evaluation. In interviews with the primary contractors responsible for developing the modernization software, the team analyzed the development processes, lifecycle, estimation techniques, system components, software development roles, and unique system components involved in an enhancement effort.

From the data collected, the team created a component-based model of the system, down to the smallest development effort. The software effort and costs were evaluated based on the components, development process used, historical productivity, and technology platforms. The result was a complete should-cost model of the entire development effort—from the top, down to the subsystem components in terms of people, processes, and technology.

The final model estimated the timing (in hours) for each phase of the software development lifecycle. It created transparency into the development process, targeted DoD areas for improvement, and streamlined cost negotiations for the modernization effort.

Results
The should-cost analysis identified a 15 to 30 percent gap between the current software development process and a targeted should-cost approach. The gap was further substantiated through a parametric model using standard software industry tools such as SEER and COCOMO.

From interviews and data audits, the team identified improvement areas, such as contractor hand-offs and inefficient testing during the integration phase, and created action items to reduce hours and costs, achieving the goals set out by the should-cost analysis. A collaborative effort between the military and its contractors successfully improved capabilities at a reduced price—providing value to all parties, including taxpayers.

Commercial projects consistently go over budget and under-deliver on functionality. Add in defense and avionics complexities, and gauging total software costs becomes quite complex. By applying standard techniques of software effort and cost modeling, an organization can accurately estimate the required work and associated cost to avoid software development overruns. This has recently been emphasized by many senior leaders across each of the services and broader DoD.

A software should-cost review is an opportunity for everyone involved to question the traditional ways of doing business and improve efficiency across the value chain. Furthermore, when targeted to a specific program’s needs, should-cost is a valuable tool for reducing costs without eroding supplier profits or cutting capabilities. It is a win-win proposition for both the DoD and its suppliers. The analysis can improve transparency and affordability by breaking the cycle of history-based cost estimation. In doing so, it answers the overriding question: What would the costs of a program be in an efficient, highly competitive environment? The sidebar on page 9, Should-Cost Modeling, discusses four elements in a should-cost analysis.
A should-cost analysis alone will not save money; rather, incorporating the results and implementing key savings initiatives bring about the savings. A successful software should-cost analysis includes a detailed action plan with specific milestones for reevaluation and an aggressive implementation mindset to prevent the should-cost from being merely a theoretical study (see sidebar: Case Study: U.S. Air Force Avionics Should-Cost on page 10).

Following are five ways to ensure that a should-cost analysis delivers accelerated, maximum benefits:

**Bring best practices to bear.** Start the should-cost analysis with an aggressive attitude for challenging the status quo. Look outside the current paradigm for best practices to replicate in a competitive environment. Focus on determining the most efficient cost-to-deliver program requirements—not on the likelihood that these requirements will be accepted. Continually ask “what if?” and “why not?”

**Perform rigorous analysis.** Acquire an in-depth understanding of the root-cost drivers and efficiency potential in major areas, including supply chain, manufacturing, program management, and overhead. Pinpoint the savings potential to support the conclusions and drive tangible actions.

**Establish the right incentives.** Recognize that the proper incentives, benefitting both the government and its suppliers, will encourage people to move beyond the status quo and act with appropriate urgency. A collaborative effort will deliver the best results. Incentives will encourage everyone to perform in a way that improves suppliers’ profits while lowering government costs.

**Translate opportunities into tangible action.** Convert cost-management opportunities into realistic action plans with clear timelines and responsibilities. Use multiple approaches to reduce costs, including negotiation, investment, joint-process improvement, and contract restructuring.

**Track performance against the cost-reduction plans.** Implement a target assurance program to identify cost-reduction targets and milestones. Review progress regularly to understand performance slips and ensure that mitigation steps are in place. Make sure progress is transparent, credible, and well managed.

A should-cost method offers a tangible value proposition for avionics software development. In fact, 15 to 45 percent of the cost of avionics development can be removed from the system by shedding light on the development process, isolating opportunities at the subcomponent level, and increasing the fact base for negotiation (see figure 5 on page 12). The cost savings are long-term because the improvements carry over multiple development cycles and modernization efforts within a program.

**The Affordable Sustainment Advantage**

Software maintenance is a growing portion of the post-development work needed to enhance and sustain weapons platforms. As software becomes more common in acquisitions, contractors’ past sustainment efforts must give way to more cost-effective and efficient government-led sustainment. With multiple service-life extension programs in effect for legacy platforms, increased government software sustainment will free contractors to focus on modernizing to keep pace with the rapid advances in sensor and weapons technology. In most instances, the government can maintain a stable sustainment organization at a considerably lower cost than primary contractors.
**In-house maintenance.** The government mandates that a significant portion of weapons systems sustainment be conducted in-house to preserve a captive capability that can be called upon in times of war. This recent legislatively enforced transition to organic sustainment makes sense and offers significant savings opportunities—for instance, the government can often provide lower wage and overhead costs in addition to payback periods of fewer than five years. Estimates for sustainment initiatives range from 15 to 30 percent in lifecycle savings over the next two decades when compared to full contractor sustainment. Additionally, annual run-rate labor savings are estimated between 30 and 40 percent following transition for representative programs (see figure 6 on page 13). Finally, through increased use of public-private partnerships, programs can anticipate further benefits to core hour requirements and capability targets while reducing risk through extended contractor support.

Organic resources are typically capable of conducting much more sophisticated sustainment activities than is often presumed. The DoD should undertake capability studies to evaluate the potential of the existing organic resources and organizational readiness to accept future workloads. Aligning modernization and software upgrade schedules should be embedded into a capability roadmap to govern overall schedules and prioritization.

---

*Source: A.T. Kearney analysis*

*United States Code, Title 10, Chapter 146, Section 2464, “Armed Forces: Core Logistics Capabilities,” page 1447,*
**Data rights acquisition.** To enable in-house software maintenance, appropriate government rights to the data are critical. Funding for data rights acquisition should be earmarked as a critical expenditure. It may be necessary to fund acquisition independently to prevent potential mingling or re-appropriation that favors imminent challenges over long-term needs. Furthermore, in-house training on the use and relevance of software data rights across program offices and the DoD will ensure that all data and data formats acquired are usable throughout the system’s life.

**Necessary skill sets.** Several key skills will be needed over the next decade. Advanced work in operational flight programs (OFPs), command and control systems (C4I), and advanced ground control stations will augment and replace workloads currently in test stand and control, and verification and validation. Software architecture and engineering skills will be required earlier in the software development lifecycle. Software sustainment will inevitably include ongoing enhancement, which requires early participation in the design process. New skills in requirements analysis, functional design, and software architecture will be required to field these new capabilities. Internally developing and supporting this work will be critical to the ongoing success of organic software sustainment organizations.

---

*This refers to command, control, communications, computers, and intelligence. Source: A.T. Kearney analysis*
By developing a targeted transformation roadmap to grow the organic sustainment organization and capabilities, the DoD can meet critical software maintenance goals that will drive next-generation weapons systems. The organic growth roadmap should prioritize the transition of avionics and C4I programs to preserve knowledge of the most critical and fastest-changing weapons system software. Transitioning these programs may result in a 30 to 40 percent run-rate savings following a transition to an 80 percent organic capability.

Transformation Is Key to Winning the Race

A radical reshaping of software policies and practices will help the U.S. military avoid falling behind in this arena. From remotely piloted aircraft and smart bombs to autonomous vehicles and advanced fighter jets, software is crucial to the success of today’s weapons systems. Focusing solely on developing and maintaining military hardware is no longer an option. With shrinking defense budgets and increasingly complex systems, the defense industry and services must fight to deliver modern software-powered weapons affordably. To deliver on this ambitious objective, the military must drastically transform its approach to software. New organizational structures, operating models, and tools will be essential to modernizing and sustaining the U.S. weapons systems. The mission is to deliver benefits today while keeping pace in the race to the future—or risk losing it.

Authors

Christian Hagen, partner, Chicago
christian.hagen@atkearney.com

Steven Hurt, partner, Dallas
steven.hurt@atkearney.com

Jeff Sorenson, partner, Washington, D.C.
jeff.sorenson@atkearney.com

Dan Wall, principal, New York
dan.wall@atkearney.com
A.T. Kearney is a global team of forward-thinking, collaborative partners that delivers immediate, meaningful results and long-term transformative advantage to clients. Since 1926, we have been trusted advisors on CEO-agenda issues to the world’s leading organizations across all major industries and sectors. A.T. Kearney’s offices are located in major business centers in 39 countries.

<table>
<thead>
<tr>
<th>Americas</th>
<th>Atlanta</th>
<th>Detroit</th>
<th>San Francisco</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Calgary</td>
<td>Houston</td>
<td>São Paulo</td>
</tr>
<tr>
<td></td>
<td>Chicago</td>
<td>Mexico City</td>
<td>Toronto</td>
</tr>
<tr>
<td></td>
<td>Dallas</td>
<td>New York</td>
<td>Washington, D.C.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Europe</th>
<th>Amsterdam</th>
<th>Istanbul</th>
<th>Oslo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Berlin</td>
<td>Kiev</td>
<td>Paris</td>
</tr>
<tr>
<td></td>
<td>Brussels</td>
<td>Lisbon</td>
<td>Prague</td>
</tr>
<tr>
<td></td>
<td>Bucharest</td>
<td>Ljubljana</td>
<td>Rome</td>
</tr>
<tr>
<td></td>
<td>Budapest</td>
<td>London</td>
<td>Stockholm</td>
</tr>
<tr>
<td></td>
<td>Copenhagen</td>
<td>Madrid</td>
<td>Stuttgart</td>
</tr>
<tr>
<td></td>
<td>Düsseldorf</td>
<td>Milan</td>
<td>Vienna</td>
</tr>
<tr>
<td></td>
<td>Frankfurt</td>
<td>Moscow</td>
<td>Warsaw</td>
</tr>
<tr>
<td></td>
<td>Helsinki</td>
<td>Munich</td>
<td>Zurich</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Asia Pacific</th>
<th>Bangkok</th>
<th>Melbourne</th>
<th>Singapore</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Beijing</td>
<td>Mumbai</td>
<td>Sydney</td>
</tr>
<tr>
<td></td>
<td>Hong Kong</td>
<td>New Delhi</td>
<td>Tokyo</td>
</tr>
<tr>
<td></td>
<td>Jakarta</td>
<td>Seoul</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kuala Lumpur</td>
<td>Shanghai</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Middle East and Africa</th>
<th>Abu Dhabi</th>
<th>Johannesburg</th>
<th>Riyadh</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dubai</td>
<td>Manama</td>
<td></td>
</tr>
</tbody>
</table>

For more information, permission to reprint or translate this work, and all other correspondence, please email: insight@atkearney.com.

A.T. Kearney Korea LLC is a separate and independent legal entity operating under the A.T. Kearney name in Korea.

© 2012, A.T. Kearney, Inc. All rights reserved.