CA Gen Best Practices Guidelines

This paper articulates CA Technologies' vision of best practices for optimal use of CA Gen within large organizations. This vision is based on CA's experience in working with large customers worldwide and supporting them in building enterprise-scale mission-critical solutions. Whilst this document outlines the high level approaches, benefits, and other considerations associated with our recommendations, it is by no means complete but forms a context for further discussion. Customers may find value assessing their current deployments against this framework, and CA would welcome the opportunity to participate in these discussions with customers.

DECEMBER, 2010

Author

CA TECHNOLOGIES
FACET CONSULTING

Table of Contents

EXECUTIVE SUMMARY 1

SECTION 1 2

CA Gen best practice framework 2

Guiding Principles of the framework 2

Five pillars of the CA Gen Best Practices Framework 3

Inter-relationships between the pillars 3

Assessing your portfolio 4

Where it matters most 4

Enterprise Application Software 4

SECTION 2 5

Enterprise Application Architecture 5

Key Application Architecture features 5

Component Based Architecture (CBD) 5

Separation of User Interface and Application Logic (n-Tiering) 6

Services Oriented Architecture (SOA) 6

Data Consolidation 6

SECTION 3 7

Development Guidelines 7

Development with CA Gen 7

Development Processes 7

Development Standards and Templates 8

Quality in Development 8

Documentation 8

Ownership 9

Change management 9

Automate Builds and Application Deployment 9

Encyclopedia management 9

SECTION 4 10

Testing Guidelines 10

Test Automation 10

Risk Based Testing 10

Component testing (CA Gen) 10

SECTION 5 11

Configuration management 11

Table of Contents

Role separation 11

Source code control 11

Model management 12

SECTION 6 13

Environment Guidelines 13

Virtualization 13

Supported software 13

The evolution of COTS 13

The benefits of COTS 13

Transparency/Performance/Governance 14

SECTION 7 15

Summary 15

APPENDIX A - CA GEN GLOBAL CUSTOMER BASE 17

Global customers by Industry sector 17

Global customers by Geography 17

APPENDIX B - CA GEN WORLDWIDE ECOSYSTEM 18

EDGE User Group 18

CA Technologies User Communities 18

CA Gen Social Networks 18

CA Gen Partner Products and Services 18

EXECUTIVE SUMMARY

Background

Periodic review of an organization's current application development practices, against a best practice framework, is a proven method of incrementally and continually improving the efficiency and effectiveness of those processes. For CA Gen customers, this paper provides a framework against which this can be done, with the objective of maximizing the value of their investment in CA Gen, and minimizing the operational costs for the CA Gen application portfolio.

Guiding principles

Three key objectives drive our vision in the use of CA Gen for the development, support and delivery of enterprise mission critical solutions for our customers:

- Minimizing the **Total Cost of Ownership** (TCO)
- Maximizing the **Productivity** of development teams
- Maximizing the **Flexibility** of Applications and Environments

Every customer's priorities, pressures and constraints across these three areas are unique, but we find these three focus areas to be shared and recurring globally.

Key recommendations

Discussion within this document identifies five critical "pillars" that underpin CA Gen best practice. The key recommendations focus on these critical pillars:

- **Enterprise Application Architecture**
  Large applications should be segmented into functional *components*
  Applications should decouple their User-Interface and Application logic (Tiering)
  New delivery channels and increased integration should be enabled by *services*
- **Development Practices**
  Define, publish and review Development processes and standards
  Automate the build and deployment of CA Gen applications
- **Testing Guidelines**
  Utilize Automation of application regression testing
  Formalize the adoption of risk-based testing methodologies
- **Change management**
  Model management strategies should support testing and deployment needs
  All application source should reside within known locations
  All application changes should be traceable from inception to delivery
- **Environment Guidelines**
  Adopt and deploy virtualized development, test and runtime environments
  Exploit Commercial-Off-The-Shelf infrastructure components where possible
  Operate development and runtime environments on supported software stacks

SECTION 1

CA Gen best practice framework

Guiding Principles of the framework

The Best Practices Framework highlights the most successful and widely adopted methodologies in CA Gen application development organizations. The framework outlined in this paper examines each of the five major pillars and relates them to the three objectives. CA Technologies has found these three principle objectives to be universally shared among our CA Gen customers operating significant application software portfolios:

1. **Minimizing Total Cost of Ownership (TCO)**

   The TCO of an application includes the initial build, ongoing maintenance costs, relevant licensing, support and operational costs over the lifetime of the application. Through this definition, minimizing costs, and maximizing the value from existing investments, are two sides of the same coin - but both are about minimizing the TCO.

2. **Maximizing Productivity**

   The Productivity of the Application Group (encompassing Architects, Analysts, Designers, Developers, Quality Assurance specialists, Technical Writers, Change management specialists, and Technical Operations staff) are one of the critical factors in the TCO determination. The Productivity of the Group is naturally determined by the efficiency and effectiveness of a wide variety of processes and procedures across the roles of the Group.

   This paper focuses the discussion on the need to achieve efficient processes in three of the most heavily utilized areas within this Group: Development, Testing, and the deployment of Change.

3. **Maximizing Flexibility**

   Flexibility within this paper encompasses two specific areas - Agility and Strategic choice.

   Agility is an area of increasing importance by our customers in delivering increasing quantities of business change to increasingly demanding timeframes. Flexibility in this context is the Application Group's capability to rapidly respond to changing requirements and reduce Time-to-Market delivery.

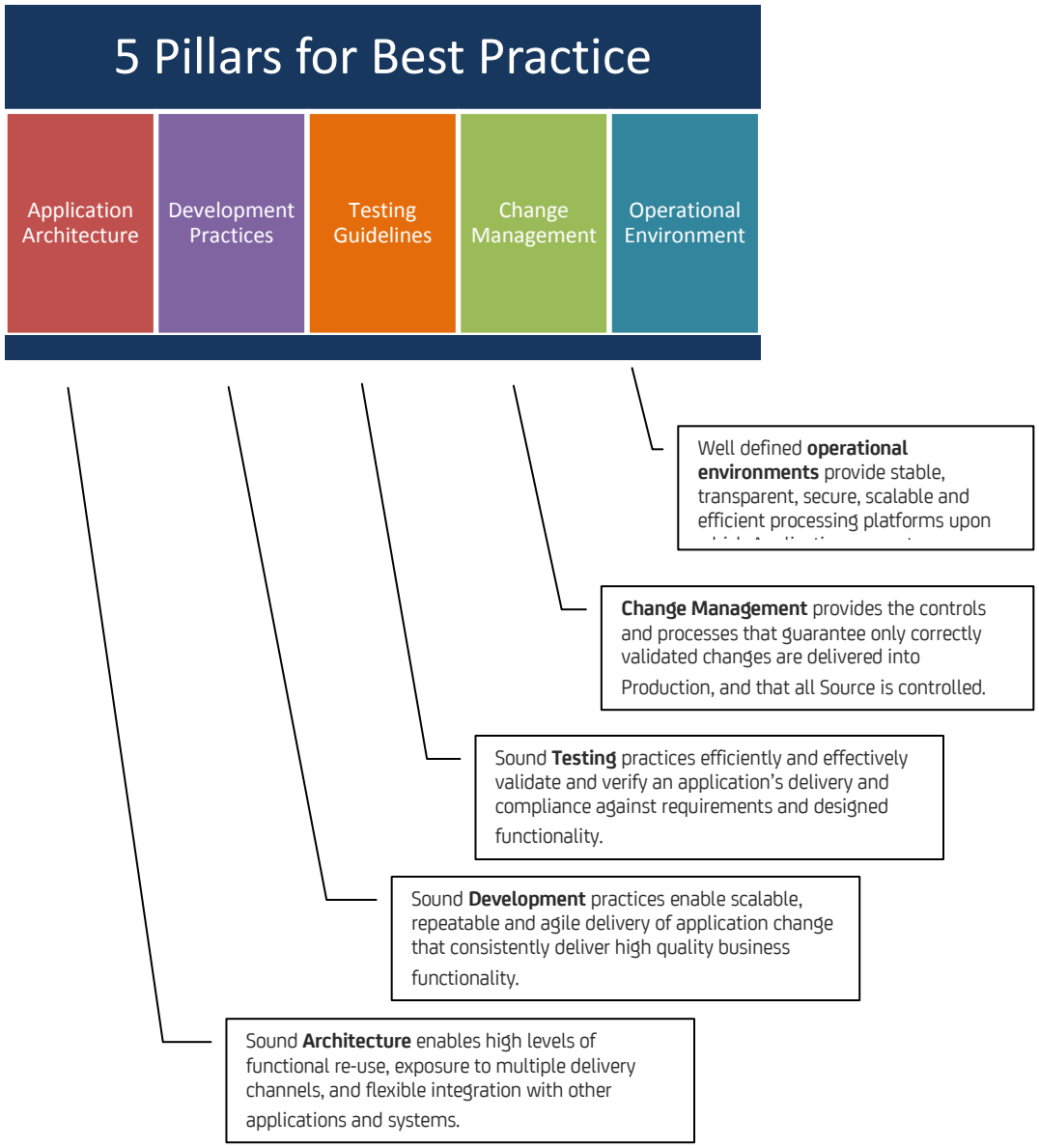
   The flexibility inherently offered by CA Gen provides valuable strategic options. This is a strong and competitive advantage to our customers, and enables them to adapt or change:

   - **Technical platforms**
     CA Gen can deploy the same application against a wide variety of technical platforms, which means CA Gen customers' applications are not inherently limited to single platforms or specific vendors. The choice of technical platform components includes operating systems, database systems, transaction monitors and underlying 3GL language.
   - **Application modality**
     CA Gen can deploy defined business functionality through a range of channels and UI's which include the traditional mainframe 3270 "green screens", rich GUI Client-Server, Browser-based delivery through Java or .NET, and also web services.

   The strategic value of this capability is that CA Gen customers can leverage the investment in their business logic, over the lifetime of those applications. This can be realized by deploying them to alternative - or additional - runtime platforms or through alternative (or new) user interfaces, as business demands require, or as cost saving opportunities arise.

Five pillars of the CA Gen Best Practices Framework

To achieve the best value from assessing application portfolios, we found that most organizations focus their attention on the five application development pillars outlined in this paper.



Inter-relationships between the pillars

Each of these five pillars has one or more effects upon another of the areas; a brief definition of these are:

1. **Application Architecture** provides the framework within which an application is designed, built and extended;
2. **Development Practices** define the processes by which those applications are built, and determine how effective and efficiently those Applications are built and subsequently maintained;
3. **Testing Guidelines** determine how the applications (or changes) are verified and validated;
4. **Change management** provides the mechanisms by which applications change - developed within the Architectural frameworks, and validated by appropriate testing, are then correctly deployed;
5. The **Operational Environment** provides that platform upon which the application executes.

Assessing your portfolio

Software systems were not created equal. Not every system in your organization is as critical as the next one. In most businesses, there are a number of systems that are more critical, core to the business and strategic to the success of an organization. These are where you need to focus your effort first.

Where it matters most

When considering the three objectives and the five pillars of best practice, every organization must assess their own unique challenges and demands.

Organizations that are in the midst of the initial design or build of a major piece of Enterprise Application Software will have a different set of priorities, from those entrusted with delivering an active change agenda to an established Application.

Equally, organizations that are challenged to move their businesses to new delivery channels, or to exploit new technology innovations, face different priorities from those charged with "keeping the lights on" with those applications that are in their twilight years.

We believe that all of the five pillars we discuss in more detail subsequently are important to differing degrees, depending on the stage of an Enterprise Application's life.

Enterprise Application Software

CA Gen is designed for the construction and delivery of Enterprise Application Software (EAS). Used by over 1,000 major public and private sector organizations across the globe, CA Gen powers mission critical applications for a range of industry sectors (see Appendix A).

So what is Enterprise Application Software?

CA Technologies defines Enterprise Application software as sharing four common features:

- Criticality - EAS is mission-critical to the Enterprise and enables significant business functionality
- Stability - EAS must be robust with applications remaining stable and available over years and decades
- Scalability - EAS supports wide-spread usage with significant transaction volumes
- Longevity - EAS operates over decades and represents a significant asset within an ICT portfolio.

These features drive management, financial, and organizational behaviors and disciplines relating to EAS.

As EAS encompasses key domain capability for an organization. As that organization changes and evolves, the EAS must consequently change - so volatility is often a consequential feature of EAS.

Whilst the lifetime management timeframe is typically long (10-30 years), ongoing demands for change require balancing the efficient and effective delivery of that business change over that period, with the need to ensure that stability is continued, scalability is retained, and longevity remains.

SECTION 2

Enterprise Application Architecture

Enterprise Application Architecture can significantly impact an organization's ability to deliver innovative, agile solutions to support business goals. In the current rapidly advancing business and technology climates, IT organizations are under increasing pressure to adapt to rapid change under strict cost and schedule limits.

A solid Enterprise Application Architecture is the foundation that can facilitate progress. Equally, poor or outmoded Application Architectures can be very real roadblocks to enabling new delivery channels, or providing significant productivity increases in the Development processes.

The following recommendations are guidelines for best practices to define enterprise application architectures that enable customers to extract the greatest value from CA Gen.

Key Application Architecture features

Customers should review their current Architectures to ensure they are delivering the best possible outcomes.

All effective Application Architectures share a set of "design" objectives that, with their adoption, should consequently deliver to the applications built against those Architectures. The key ones are these:

- **Consistent structure**
  For applications that exist for decades, consistent internal structure is a significant issue that directly affects the subsequent maintenance costs for those applications. Applications that are clearly and consistently structured enable supporting application developers to work on the content of the problem, rather than on the structure of the problem;
- **Loose coupling**
  Coupling is the degree to which modules are dependent upon one another; Applications that are loosely coupled tend to be easier and cheaper to maintain because a change in one module tends to restrict the impact of that change to that module. Applications that are tightly coupled result in changes in one module rippling widely with unforeseen consequences.
- **High cohesion**
  Cohesion is the degree to which functionally related modules reside and operate together. Highly cohesive application systems are designed to be structured so that cohesion is high by design and not by accident.
- **User interface and Logic separation**
  Critical for long-term application flexibility, strong application architectures will ensure that the investment in encapsulating business rules and data in large application systems can be exploited simultaneously by a variety of User Interfaces.
- **Data Consolidation**
  The information an application can discern from data is key in empowering informed business decisions. Over time database structures often mushroom and warrant review.

Component-based Architectures, as defined in the Component-Based Development (CBD) approach, Services Oriented Architectures (SOA), and n-Tiering are as successful and widely adopted as they are because they are all proven to deliver these benefits. Each is discussed in more detail below.

To understand how each of these three fit together, consider by contrast a single large monolithic application residing within a single CA Gen model:

- Componentization using CBD, would slice this application up vertically, separating the monolith into a number of smaller functional slices;
- n-Tiering, by contrast, would slice that application up horizontally separating out the data access at the bottom, from the Business logic in the middle, and the User-Interface layer at the top;
- SOA, would then expose that business logic as a set of defined and published services that could be used by any number of authorized internal or external application consumers.

It is important to understand that each of these three major Architectural frameworks bring something significant to enterprise scale application design and development. Each on their own deliver significant value, but very large applications, and particularly volatile ones, are likely to require all three to remain manageable, agile, and effective.

Component Based Architecture (CBD)

Customers with very large application models should consider segmenting those models into Components.

Building your applications from reusable well-defined components is a sound foundation for good architecture practices. It will allow you to accelerate feature delivery, extend the capabilities of your systems and better adapt to changes in business needs.

CA Technologies have defined a significant body of knowledge on the use of Component Based Development within CA Gen, and there exists significant support for the use of Components within the CA Gen product set. There are available utility components that can assist the rapid development of new application systems by exploiting existing pre-built components.

The key features of components are the encapsulation of the business activities (operations), with the data and the access to the data. These operations are then published with their interface signatures so that external consumers can confidently utilize them without further knowledge of how their published functionality is delivered.

This ensures that components separate out both the functional process and the underlying data, from large applications - meaning that large applications can be effectively segmented. This in turn leads to improved maintainability because the coupling is very low, but the cohesion delivered is very high - key architectural objectives.

Practically this means that changes made to components are unlikely to ripple beyond that component, so consequently they can be changed and tested more confidently.

Separation of User Interface and Application Logic (n-Tiering)

Customers should adopt Application Architectures that separate the User Interface from the Application Logic.

Application logic (business rules) should be independent from the way the application presents and collects its data. You should be able to isolate the business logic in separate action blocks that fulfill specific business functions. You should be able to defer the decision of how you will present the data to another layer that you can swap out based on changes in the application environment; as an example, you can have the same business functionality in an action block being presented through a 3270 screen, a thick client, a web interface, and on a mobile device.

Services Oriented Architecture, and similarly CBD, or other structures can all deliver this powerful outcome.

Customers with applications that have the User Interface and the Application Logic consolidated may not be able to easily adopt new delivery channels for their application solutions. There are tools and solutions available to help transition to new Architectures.

Services Oriented Architecture (SOA)

Customers with applications that require functional delivery over multiple concurrent channels should adopt SOA.

Applications already segmented into Components can have key business functions defined and delivered as services from CA Gen. This enables a single codebase to be delivered out through multiple user interface channels, whilst retaining a single implementation of the common underlying business rules.

CA Gen also includes the inbuilt capability to expose defined services as Web Services, and within a CA Gen application to consume other (CA Gen and non-CA Gen) Web Services. This powerful capability enables CA Gen applications to provide application integration capability where large portfolios of disparate software are utilized to deliver focused business functions to new channels, new markets, or with new technologies.

SOA is a long-proven capability within CA Gen technology and application systems, and some of these are quoted in Industry Analyst whitepapers as "best practice" implementations within a global context.

Data Consolidation

Customers should understand the underlying sources of data for applications within their portfolios.

Business growth forces IT divisions within an organization to evolve with changing business demands. Applications are introduced to support new requirements. Existing information is processed and analyzed in new ways to gain more insight into critical business challenges. Companies merge and acquire other entities, further accelerating their growth into new areas.

Unfortunately, the data landscape does not always evolve in a strictly controlled and organized manner. Very often redundant and inconsistent information exists where the same data is represented in many different ways by different applications. Inevitably, applications draw data from a wide range of diverse systems that each holds different pieces of the required information.

Reviewing the most critical data to the business and devising transformations to reach the most efficient representation should be carefully investigated. Effective data representation and usage can dramatically improve the efficiency of applications, but more importantly lead to data consolidation and the eventual removal of duplicated (redundant) data.

SECTION 3

Development Guidelines

Development with CA Gen

CA Gen is a model-driven development platform that encompasses both logical and physical modeling, for both the organization's data model, and the processes that utilize it. The central repository that holds all the CA Gen models (the Encyclopedia), by its integrated nature, enables thorough and complete impact analysis on any proposed change for an application.

It is because of these key aspects of CA Gen that many of the typically sought features of large-scale applications are inherently and automatically delivered by CA Gen. These include:

- Data-centric design and modeling
- Process and data independence
- Logical and Physical design separation
- Technical platform independence

Despite these significant advantages over 3GL development technologies, the basics of Enterprise scale application development continue to apply equally:

Development Processes

Organizations should have defined development processes appropriate to their Development objectives.

Organizations may have a set of alternative development processes that are appropriate in different circumstances, types of projects, or perhaps specific technologies. All of them share the objective of ensuring that the appropriate level of controls to deliver a high-quality outcome, within the defined project constraints, are present.

Well-established software development methodologies are abundant. Different development organization cultures and the nature of the work targeted can be more suited for one methodology over the other. Agile methodologies such as Scrum are on the rise due to their ability to adapt to changing requirements and most cost effective to change.

The development processes cover the gathering of requirements, the subsequent phases of analysis, design, development and testing, and the various milestones, checkpoints, and controls. Development processes have a large number of audiences which include:

- **Analysts, Developers and Testers**
  use the process selected for them by their Project Manager, as the roadmap for their major activities, and to understand how and where they work with others throughout the project
- **Team Leaders and Project Managers**
  use the process to ensure that the appropriate resource types are available through the process phases, and that the right checkpoints, sign-offs, and milestones are achieved and monitored
- **Resource Planners and Financial Controllers**
  use the process to understand where specific financial milestones within the process lie, and what resources are involved when, for what duration, and to understand the cost and resource profile
- **IT Auditors**
  use the process as a benchmark to assess actual activity against, and where divergences are identified, to then determine if the process should be modified or guidance issued - or both.

The definition of the Development process is the most fundamental step in building IT Development maturity as assessed against the *Software Capability Maturity Model (CMM)* from Carnegie Mellon University.

Development Standards and Templates

Organizations should have defined Standards that support their Architecture, and templates that support their Development and Change management processes.

Within the Development process, Application standards are a critical feature for consistently delivering high-quality applications with a consistent structure and performance. At the most basic level, the Application standards are the coding-level definition of the defined Application Architectures for an Application. Due to this, there should be clear references from the standards, back to the published Architecture documents so that Developers can quickly understand the context and reasons for the standards' existence.

It is important to remember that Application standards define the way to solve the problem; they do not ensure that the problem is solved correctly. Application development standards usually focus on coding standards, but should also include data modeling standards, which are critical to Enterprise-scale applications.

Equally aligned with getting consistent outcomes are a set of appropriate templates for each of the roles and phases throughout the development process. These include costing and planning templates, specification and design templates, coding templates and examples, unit and system testing templates, and migration and deployment templates. By ensuring the templates within a process mesh with the process (key information, audience, structure etc), all of the Development group get to focus on the business problem they are solving, rather than the mechanics and interactions required to do the solving.

The established standards need to be well understood and must be enforced.

Basic training needs to be periodically offered to your staff to ensure the standards are well understood, and equally to encourage feedback from those charged with delivering, which areas require improvement.

Quality in Development

As implied in the previous sections relating to Development processes and standards, quality in the resulting application software is a key objective. The key message is of course that quality is cheapest to design in, rather than retrospectively test in.

Where quality is a design metric for the Development Standards and Processes, and those are enforced, then higher quality outcomes will be delivered at the lowest cost. As a multitude of studies have shown, the later defects are identified in a development cycle, the higher the rectification cost is.

Processes that define the entry and exit point artifacts, with supporting templates that show how those artifacts are constructed, provide quality that is built into the process. Quality checkpoints throughout the development lifecycle are a key feature of quality-focused software development processes.

Documentation

Your application systems must be documented and these artifacts must be stored and accessible.

The type and level of documentation required should be a function of your process definitions - and the key factor is that all documentation must serve a defined purpose.

There are two main categories of documentation for long-lived enterprise-scale applications software:

- **Application documents (Evergreen)**
  These are the documents that exist for the lifetime of the application, and are kept current throughout its life, because they are used as a reference to build, enhance and extend the application. Examples of these documents include Architecture, Standards, Business Requirements, Functional definitions, Component specifications, Testing packs, User documentation, Training material.
- **Project documents (Point-in-time)**
  These documents are equally as crucial as the Application documents, but exist for a specific timeframe only - after which they are archived as historical references for audit and compliance purposes only. Project plans, budgets and schedules, along with coding specifications and change specifications are all examples of artifacts that should be archived once a project completes.

It is important to ensure that there is a shared understanding about which documents fit into which category so that they are controlled, published and made accessible appropriately. Most project activity will almost always amend one or more of the Application documents during a Project lifecycle.

Ownership

The ownership (Technical and Business) should be nominated and published for all significant applications.

There are several types of ownership of enterprise applications, but at a minimum there are two that we see consistently and valuably identified across much of our customer base:

- **Business ownership**
  The Business owner of an Application system is usually the nominated individual within the Business area that is principally served by the application system. Ultimate responsibility for determining what change is approved to proceed, if the tested quality levels are acceptable, and what levels of resourcing are appropriate below to this type of ownership.
- **Technical ownership**
  The Technical owner of an application system is usually the Development Manager, Architect, Lead developer or senior analyst for that application that has a significant and detailed technical understanding of the application scope and function. The Technical owner vets all the major design decisions for the application to ensure the application remains consistently structured and complies with defined Architecture.

Successfully managed application systems usually have positive and close relationships between these two types of owners - this delivers shared understanding and appreciation for the objectives each needs to achieve for both the short-term and long-term benefits of the organization.

Change management

The Change Management processes should be clearly defined, and should integrate with the Development processes.

Change is an integral part of every software development process. You must have an established process for initiating, accepting and testing changes to your application system. For many industry sectors this is not just a requirement of best - or even just good - practice, but one mandated by law with significant penalties for non-compliance.

The Change Management process defines how change is initiated - from whatever source, then how those changes are assessed and prioritized. Subsequently they are either rejected, accepted, or deferred for subsequent action. The Change process also extends through to the delivery of that change, and the eventual notification to the originally initiating party advising that the requested change has been deployed.

The Change management process sits around the Development process, but clearly must integrate with it at key points. The specific roles, delegations, authorities and responsibilities are key definitions within the Change Management process.

Automate Builds and Application Deployment

Application build activity (3GL code generation, compilation, linking and DB binding), and the subsequent deployment of the artifacts, should be automated wherever possible.

The Build and Deployment activity within the Development process is usually the most frequently utilized one, and so consequently benefits the most from automation. The benefits are realized in two distinct ways:

- **Faster time to market**
  With a Build and Deployment process automated, the time to delivery is immediately reduced because the most frequently utilized activity is shortened in duration;
- **Removal of manual errors**
  By automating this process, the potential risk of failing to correctly Build or to Deploy software components is removed. This directly results in an increase in application quality.

There are a range of third-party tools that can assist CA Gen customers achieve this outcome, and for those customers yet to start down this path, there are significant benefits to be obtained.

Achievement of this capability for a full application rebuild, in combination with automated regression testing, can be the basis for regularly rebuilding and retesting the entire, or large portions, of application systems.

Encyclopedia management

Customers should ensure that their CA Gen Encyclopedia is managed as a critical piece of infrastructure.

The CA Gen Encyclopedia is the central code repository for all CA Gen models, and the central and critical piece of the development framework. Extraneous models should be archived and removed, the underlying database tuned, access carefully controlled, appropriate product patches promptly applied, and availability maintained. Poorly managed encyclopedias result in slow development processes, unnecessary delays to almost every development activity and a consequential reduction in productivity for the entire development group.

SECTION 4

Testing Guidelines

Test Automation

Customers should consider automating load and performance testing of their applications.

There are many tools that can help you automate functional, load and performance testing; the principle advantage from automated testing is the dramatic reduction in the duration required to execute a set of test plans. As testing represents a very significant proportion of the enterprise application development cost - both in effort and duration - the delivery of automated testing can deliver significant inroads to reducing time-to-market for the overall development process.

Customers should expect that the journey to deliver this outcome will not be simple, and that the larger and more complex the application is, the more difficult this may be. This is particularly so with large application systems with multitudes of interfaces to external applications or organizations, or applications that deal with time-sensitive data from external sources.

We typically see that the testing costs and duration represent between 25% and 50% of the total development process, so the potential gains from delivering automated testing are potentially very significant.

Risk Based Testing

Customers should formally assess the risk levels of their applications and application components.

From a technical perspective, analyze your application usage to understand volumes of transactions, and throughout the year, if there are particular "hotspots". Common components and load modules that are most highly used should be assigned higher priority than less critical areas of your applications.

Equally, from a business perspective, certain business processes will be deemed critical to business continuity, so the transactions that underpin these critical processes should also be assigned higher priority than other parts of the applications.

Where testing resources are limited, focus your testing on higher priority test cases; where resource constraints are not so pervasive, the higher priority areas should still remain the initial focus.

Component testing (CA Gen)

Customers utilizing CBD should consider adopting specific CBD testing strategies.

For customers that adopt Component Based Development (CBD), there can be significant value (reduced time-to-market & increased quality) in utilizing specific testing strategies that focus on functional Components. Although it is possible to include the User Interface (UI) definition within CBD boundaries, the vast majority of our customers' components deliver published technical interfaces without UI; this commentary assumes this case.

Remembering that CBD delivers neatly segmented functional parts of major application systems, testing strategies for components can exploit this. Specifically, (automated) testing can focus on exercising each of the functional component's published "operations" meaning that each component can be tested independently of the wider application functionality which consumes the component.

Where customers are considering the introduction of automated testing tools and approaches, the adoption of this approach can be a valuable and low-risk method of delivering value, learning how best to utilize the testing technologies selected, and yet not upend the existing approaches.

For application systems that make extensive use of components, this can potentially have a major impact upon the entire application testing strategy. If, for example, 70% of an Enterprise Application was delivered using components, and components were tested independently, then the overall risk and coverage required from the integrated application testing is dramatically reduced.

Without CBD (or SOA), functional testing of an application must use an exhaustive set of tests to exercise each application function through use of the applications User Interface. In these applications, all of the application code resides within the application itself (there are no isolated components). The larger the application grows, the more expensive this becomes.

The volume of tests required to cover an application increases exponentially with the size of the application - not linearly. This is because of the "interactions" - or the level of coupling - between the internal parts of the application, increases far beyond the rate of increase in the size of the application.

With CBD, each component is a separate and isolated unit that may be tested as a stand-alone "mini-application". Inherent in the definition of a component is the explicitly defined functional operation of that component, and exactly what each operation of the component performs.

SECTION 5

Configuration management

Customers should ensure that their Configuration Management is defined, implemented and complete.

Configuration management is the set of processes and supporting tools that deliver control over all of the artifacts of your development process. Specifically, each of these must be controlled, re-creatable, traceable and archived.

- **Controlled environment**
  you must have well established levels of authority for every individual's role within your organization
- **Re-creatable**
  you must have a well-established control system to enable you to recreate a snap shot of the generated, and compiled artifacts, and the source code at a specific previous point-in-time
- **Traceable**
  you must have policies and procedures in place to enable you to trace changes to team members and relate them to the cause of a change e.g. defect resolution or enhancement request
- **Archives**
  Formal deliverables must be archived in their entirety to allow auditing and disaster recovery.

Taken within the context of the wider Change Management (discussed in the preceding section), Configuration Management is a key part of that process and focuses on the delivery of the technical artifacts.

Role separation

Customers must ensure complete role separation between Development and Configuration Management groups.

The separation of roles within the Development process is of particular interest to IT Auditors, and is especially crucial with Configuration Management. Specifically, the staff responsible for the deployment of change from Development into Controlled environments (formal testing and eventually production), should always be separate from development staff.

This role separation ensures that developers can never "release" modified code into production without passing through the control gate that is Configuration Management. This role separation should be enforced through the security policies and access controls to environments, libraries, databases and models.

Source code control

Customers must ensure that all technical artifacts necessary to build and deploy applications are under source code control.

For CA Gen applications, there are several components that constitute your source code outside of CA Gen models (discussed subsequently under *Model Management*):

- The 3GL source code generated from the model
- External Action Blocks and User Exits
- Bitmaps, OCX controls, ASP.Net web controls, etc
- Command scripts, Rexx, JCL, DDL, SQL

These components have to be versioned and archived in a source control system. You must be able to recreate any version of your application from these artifacts.

From the wider application perspective, there should be a central and controlled repository for application source artifacts, irrespective of the technology involved - Java, "C", COBOL, JCL .

The critical element is that all the technical artifacts necessary to build and deploy an executing application system in production should be under source code control.

There are a wide variety of Source code control products available, but two critical features that need to be implemented are:

- Mandatory and consistent use of the supported tools (and processes) across the organization.
  In particular, avoid individual application teams each using their own favorite control product; In a large part the choice of the product is less important than the fact that it needs to be universally deployed, mandated and utilized;
- Administration and usage of the tools by *Configuration Management* staff who have no overlap with development roles.

Model management

Customers must define, publish, execute and enforce appropriate Model management strategies for their CA Gen applications.

For CA Gen, models are the true source for CA Gen applications, so consequently must be carefully managed and controlled. The objectives of an effective Model Management strategy are to ensure that:

- The models necessary to control, trace and (re)create development, test and production deployments are maintained
- The models necessary to support the required environments, streams, and applications are in place
- The segmentation of control correctly restricts appropriate roles to the models they need access to
- Appropriate models are available to support emergency Production "hotfixes"
- The strategy supports the scale and streams of development work expected for each Application
- Controlled models supporting controlled environments are appropriately quarantined and protected.

Good model management practices have to be enforced, and provide the fundamental separation between the Development community, the Testing community, the controlled environments, and ultimately Production.

Model management strategies are not necessarily set in stone for any organization. Customers undergoing significant change where the level of application volatility drastically increases or decreases often warrant a review of the current strategies.

Strategies in place that work well for small teams of Developers delivering low quantities of change, usually do not scale well if the volume of change or quantity of Developers drastically increases.

Conversely, sophisticated strategies designed to deal with parallel development streams for multiple development teams in alternate geographies and time-zones, can be expensive overkill if there is a dramatic reduction in the volume of change and applications move to a "lights-on only" mode.

At an absolute minimum, there will be two "levels" of CA Gen models for any application system. The first supports the development environment for the application system, whilst the second, which is derived under controlled migration from the first, supports the Production application. There are many additional layers, streams, and sets of models potentially involved, but these are the two absolutely basic and mandatory ones. Customers that do not meet this should immediately review their current Model Management strategies and consider changes to it.

SECTION 6

Environment Guidelines

Virtualization

Customers deploying applications to distributed environments should consider adopting virtualization for those environments to improve development and testing time-to-market benefits, and operating cost reductions.

Increasingly, organizations recognize the value of using virtual environments. There are significant cost savings in hardware acquisition costs. However, in our experience, the major benefits come from resource and application management and maintenance savings.

Virtualization can be applied to the application runtime environments - especially where the application servers are deployed on the distributed platforms (Microsoft, Linux, or Unix variants). As a function of virtualizing the application runtime environments, both the testing and development activities can exploit these benefits.

Specifically, with virtualized environments, new development and testing environments can be almost-instantly configured and deployed. This can deliver dramatic improvements in the provisioning time for environments and in the "known-state" stability of environments during the development lifecycle.

Supported software

Ensure that 3rd party software has active support agreements with appropriate SLAs so that software fixes can be obtained when necessary. This is particularly important for mission-critical systems where Business continuity may be affected by prolonged application outages.

Wherever possible, use Commercial Off the Shelf Solutions (COTS) in conjunction with your CA Gen tools. Examples include security management, source code control systems, application build environments, print and application servers, and test automation tools.

Whilst developing in-house custom applications may provide a short-term solution to satisfy an immediate need, it has long term maintenance ramifications; these can range from the ongoing need for dedicated resources for support and maintenance, to the inability to subsequently take advantage of newer industry standard features.

The need for supported software applies equally to Open-source software where it has been utilized within your software stack solution. The critical - and often overlooked - requirement for major enterprises is that the software will be supported for the long term - because large organizations know their applications will be in place for the long term.

The evolution of COTS

One of the primary and underlying themes of change within the ICT landscape has been commoditization.

In the application software space, as similar organizational challenges are solved repeatedly, the wider industry recognizes common solution patterns, and from this arises commercial software offerings. These offerings are typically customizable and generic, and support a very wide-range of their specific solution target market. Software solutions from Seibel for Customer Relationship Management (CRM), SAP for Enterprise Resource Planning (ERP), and SAGE General Accounting.

In the hardware space, virtualization has enabled the separation of the application and infrastructure software platforms, from the specific hardware that they execute on. Hardware resources such as levels of processing power, quantities of memory, pools of storage, and volumes of bandwidth, can now be provisioned, managed, and changed independently of the application software stacks that execute upon then. The emergence of Cloud-based offerings have accelerated this separation, and dramatically accelerated the view that hardware resources are now real commodity items.

In the infrastructure software space, the same outcome has arisen with a vast array of infrastructure software products available in the market today. These cover Performance and Governance, Security, Authentication and Anti-virus, Print, Application, and Web servers, and a vast array of products to support, control, and delivery software development activity.

This commoditization activity has given rise to COTS - Commercial Off-The-Shelf software.

The benefits of COTS

There are four primary benefits from utilizing Commercial Off-The-Shelf software:

- **Focus on Business IP**
  Customers that exploit commoditized software offerings no longer have to invest precious resources in the creation and support of such functionality. Those customers can invest solely in the unique and domain-specific software that powers their own business - providing efficient and effective delivery, and a competitive and market leading edge to the overall organization;

- **Risk reduction**
  Customers that replace in-house developed solutions, for what are now commoditized software products, effectively out-source risk from themselves, to the supporting Vendor. On the assumption that the commoditized software offering is provisioned with Vendor support, then the Vendor assumes the risk for that component of the Organization's software stack - both its functional operation and its ongoing support and future enhancement;
- **Simplification**
  Customers that utilize commodity components within their software stacks are usually able to simplify their overall software portfolios by virtue of the generic structure, robust and proven solutions, and the ability to call on industry-standard resource service offerings to support those products where required;
- **Cost savings**
  Customers often also benefit from long-term recurrent operational cost savings because the sophistication of functionality offered by proven commercial products can rarely be provided within internal budget constraints. Global software vendors also have the scale and competitive drive to ensure that ongoing enhancement retains their market edge, and that pricing remains competitive.

In recognizing the clear benefits from the adoption of commoditized software solutions, it is important to remember that in-house delivered functionality, within existing software stacks, will usually have been built for very solid reasons. The principal "take-away" is that as time moves forward, there will be natural decision points within an application's lifecycle, or an Organization's, where there can be significant value delivered by assessing and potentially replacing these components.

Transparency/Performance/Governance

You should have procedures and supporting tools in place that provide monitoring and controlling of your application development lifecycle, your deployment environments, and the executing applications.

These should be well-known to the development teams and the auditing bodies within your group.

With the monitoring of the application development processes, the state and stage of every application should be publicly known across the entire group. This becomes especially important when multiple development teams and multiple development and testing environments are in use.

The monitoring of the actual environments is again crucial - both during the development cycle when there are multiple environments, teams, databases and applications all in use. Equally in the production environment, having the status of all relevant environments actively and automatically monitored provides basic service assurance.

Lastly, the monitoring of executing applications is important for two different reasons:

- **Assurance**
  With executing applications monitored from the Production environments, all key stakeholders (technical and business) can be satisfied that the applications are operating within agreed service levels. Further, if real-time monitoring is in place, then that assurance can be extended into proactively controlling information and necessary action when applications, or parts of them, fail;
- **Performance**
  As applications expand in their functional growth through increased functional requirements, or their usage extends through uptakes in usage, the performance of the application within the key environments becomes keenly observed. The capability to identify key performance hotspots within an application, and then proactively manage them can not only deliver an improved user experience, but also direct cost benefits by the deferral of infrastructure upgrades (distributed platforms), or the reduction in MIPS utilization (mainframe).

SECTION 7

Summary

CA Gen Model Based Development shields applications from changes in target environments and platforms. Assessing your current development practices against the areas and recommendations outlined in this Whitepaper can help you extract more value from your investment, reduce your operational costs, increase your productivity and throughput, and improve both agility and the quality of your solutions. The following table summarizes the key areas within each of the five pillars and matches them back to the three governing objectives as a quick assessment tool:

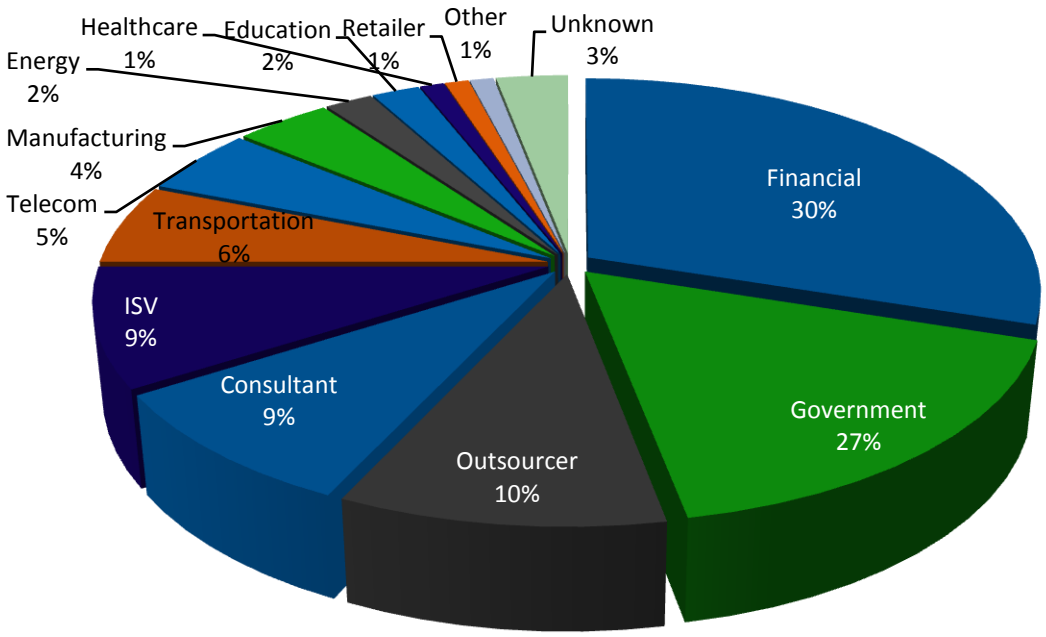
	REDUCES TCO	INCREASES PRODUCTIVITY	IMPROVES FLEXIBILITY
ENTERPRISE APPLICATION ARCHITECTURE	✓	✓	✓
1. CBD segments large applications so that development , testing and re-use becomes more effective; this **increases productivity**			
2. n-Tiering segments large applications into layers which **maximizes flexibility** and **reduces TCO** through improved maintenance practices			
3. SOA enables high levels of business functional re-use, which **increases productivity** and **maximizes flexibility**			
4. SOA enables exposure of existing business functions through multiple channels which **maximizes flexibility**			
5. Data Consolidation enables removal of redundant data sources and allows application consolidation which **reduces TCO**			
DEVELOPMENT PRACTICES	✓	✓	✓
1. Defined, published and enforced development processes **increase productivity**			
2. Defined, published and enforced standards that support Architectures **increase productivity,** and **reduce TCO**			
3. Quality objectives are designed into processes, standards and templates to **increase productivity** and **reduce TCO**			
4. Key documentation artifacts are created and maintained to **maximize flexibility** and **reduce TCO**			
5. Business and Technical owners are identified and accountable which **reduces TCO**			
6. Change Management is defined and implemented and directly **reduces TCO** by ensuring only relevant changes are progressed			
7. Automated application build and deployment directly **increases productivity**, **reduces TCO**, and **maximizes flexibility**			
8. Effective Encyclopedia management ensures key infrastructure is effective and so **reduces TCO**			
TESTING GUIDELINES	✓	✓	✓
1. Automated testing directly reduces time to market which **increases productivity** and **reduces TCO**			

	REDUCES TCO	INCREASES PRODUCTIVITY	IMPROVES FLEXIBILITY
2. Risk based testing ensures that effort is always prioritized and focused correctly which **increases productivity**			
3. Component testing can isolate significant portions of application functionality from UI based testing which **increases productivity** and **reduces TCO**			
CONFIGURATION MANAGEMENT	✓	✓	✓
1. Role definition and separation for Configuration Management **increases productivity** and **reduces TCO** through risk control			
2. Source code control provides key management of critical source which **increases productivity** and **reduces TCO**			
3. Effective model management underpins development co-ordination and so **increases productivity, reduces TCO** and **delivers flexibility** through the capability to easily and quickly deploy application change.			
ENVIRONMENTAL GUIDELINES	✓	✓	
1. Virtualization of environments enables rapid environmental instantiation which **increases productivity** and **reduces TCO**			
2. Supported software directly **reduces TCO** by ensuring vendors hold the risk on software failure			
3. Utilization of COTS over in-house software **increases productivity** due to focus on business relevant IP investment in software			
4. Active monitoring of application execution provides Governance and performance opportunities which **reduce TCO**			

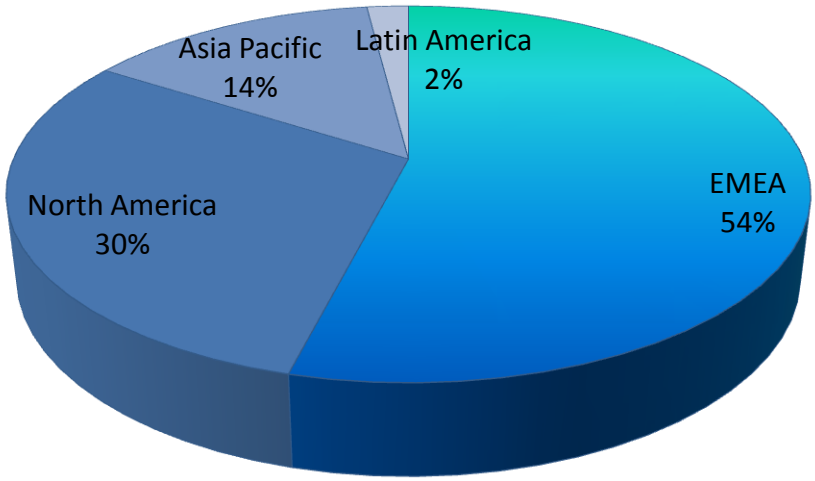
APPENDIX A - CA GEN GLOBAL CUSTOMER BASE

Hundreds of businesses and governments around the globe rely on CA Gen to deliver and maintain their most strategic enterprise applications.

Global customers by Industry sector



Global customers by Geography



APPENDIX B - CA GEN WORLDWIDE ECOSYSTEM

EDGE User Group

CA Gen customers have an active user group called EDGE. Global user group conferences are held throughout the world. Please visit http://edgeusergroup.org.

CA Technologies User Communities

In addition to the EDGE website, customers can collaborate on CA Gen User Community websites for additional information, best practices and resources. Please visit http://ca.com/communities (search for "Gen").

CA Gen Social Networks

CA Gen has a large following on many social networks including:

http://twitter.com/ca_gen

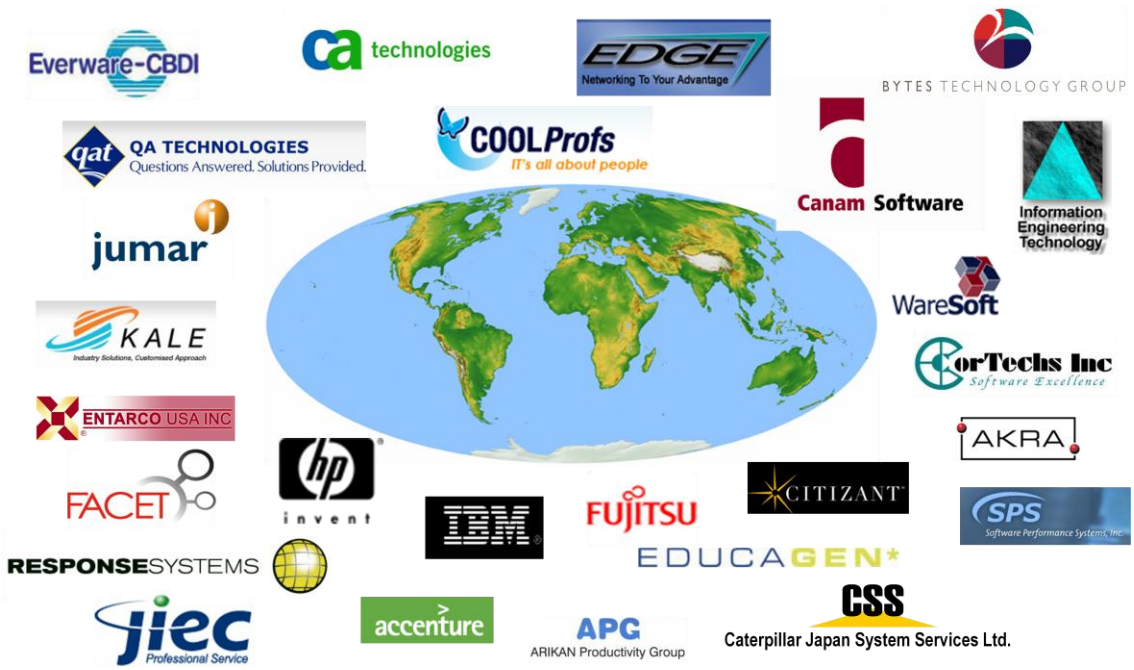
http://linkedin.com/

http://gentalk.biz

http://duick.com

http://flickr.com/_(ca_gen)

CA Gen Partner Products and Services



CA Gen relies on a strong worldwide partner ecosystem to provide services and complementary solutions for our customers. A listing of partner products as well as partners who offer services can be found on these websites:

http://ca.com/gen

http://www..ca.com/communities (search for "Gen")