

How to Avoid an IAM Trainwreck

Dave Nesbitt, Identity Architect, Oxford Computer Group

Introduction

Identity and Access Management (IAM) is a good thing. Most people know this by now. Most people realize that any organization which manages a reasonable amount of users, whether these are employees, customers or other user types, needs a n IAM strategy and some IAM infrastructure to make the job easier. But for some reason, some people seem to think it's a hard thing to do and that IAM projects are costly, long-winded and drawn out - in other words, trainwrecks.

We don't understand this. The IAM projects we get involved in (and as that's all we do, that's quite a few) tend to take no more than 3 months and rarely run much over six figures – including hardware, software and professional services. So we've put together a few hints and tips we've picked up on the way that should help you avoid some of the common pitfalls that can turn your IAM project into a disaster.

Overview

Just in case you haven't yet heard the IAM message, or need a quick reminder, here's a quick overview. Identity and Access Management (IAM) is defined by Kim Cameron, Identity Architect at Microsoft, as "*a system of procedures, policies and technologies to manage the lifecycle and entitlements of digital identities*". At Oxford we haven't yet found a better definition and so have adopted this as our preferred starting point for any deeper discussion of IAM.

A *digital identity* is a set of data that makes *claims* about a person (or computer, device or other digital resource). These claims could simply describe the subject (name, email address or age etc.) or they could also be claims to access rights to protected resources. IAM, then, is procedures, policies and technology that manage the *lifecycle* and *entitlements* of these digital identities. As you and I are technologists, then of course it's the technology we're primarily interested in here, so in practical terms, we're talking about user objects in databases or directories that are used by applications to control access rights.

The problem is that these users have proliferated in recent years, mainly caused by the widespread adoption of client-server and Internet-enabled technology. Typically, each identity-aware application is deployed with its own identity database, which can cause the following problems to occur:

- **Large Administrative Burden** – the burden on the helpdesk becomes enormous. New user accounts are created and maintained manually in each data store and can take days to become active.
- **Poor User Experience** – users are often required to remember many different usernames and passwords. This often leads to forgotten passwords and more calls to the helpdesk to ask for password resets.
- **High Security Risks** – users are often given incorrect access rights, and rights tend to accrue over time. Old accounts are not deleted or deprovisioned, even after users have long departed the organization

In response to this, software vendors developed IAM products designed to help organizations to overcome these problems and realize the significant business benefits that a coherent IAM strategy and architecture can bring. Products such as

metadirectories, provisioning platforms, access control platforms, delegated administration interfaces, workflow and password management can be found amongst the wide mix of offerings from vendors.

Trainwrecks and How to Avoid Them

Defining your requirements, picking the right products and designing an IAM architecture to meet these requirements is a fine art, and one that we don't have space to address adequately in this article, so let's assume that you've done this already. However, things haven't gone exactly as you planned. You're six months into the project, progress has been patchy to put it kindly and the project board are breathing down your neck looking for results. Where did it all go wrong? How did your project come off the rails and become another IAM trainwreck?

We think that there are three common types of IAM trainwrecks that can occur. But we also think we know why they occur and how to avoid them. Here goes...

Type 1 – The Head On Collision

When your project comes to a sudden, jolting and painful stop, you've suffered a head-on collision. Either another project is in your way, or someone is obstructing you. A common project collision occurs when the HR team, unbeknown to you, decide to implement their own IAM application – perhaps a user self-service application has been implemented as a bolt-on to their existing HR database. Before you know it your killer application is superfluous, or HR are refusing to provide you with a data feed of all employees because they don't think you need it. The other typical obstruction occurs when someone is actively obstructing your progress, perhaps because your flashy new application is about to make their handcrafted COBOL provisioning system redundant after 20 years. Would you help someone who was trying to put you out of a job? IAM projects touch a lot of other systems, not all of which want to be touched.

The solution to both these problem is the same – secure executive sponsorship before you start. Many IAM projects fail to go high enough up the organization in search of a project sponsor. We recommend you go as high as you can. Having an executive sponsor on your side will win hearts and minds across the board and, if all else fails, intimidate those who are trying to obstruct you. How do you get an executive sponsor? Simple – before you start write a cast-iron business case for the project. Remember the four benefits of IAM:

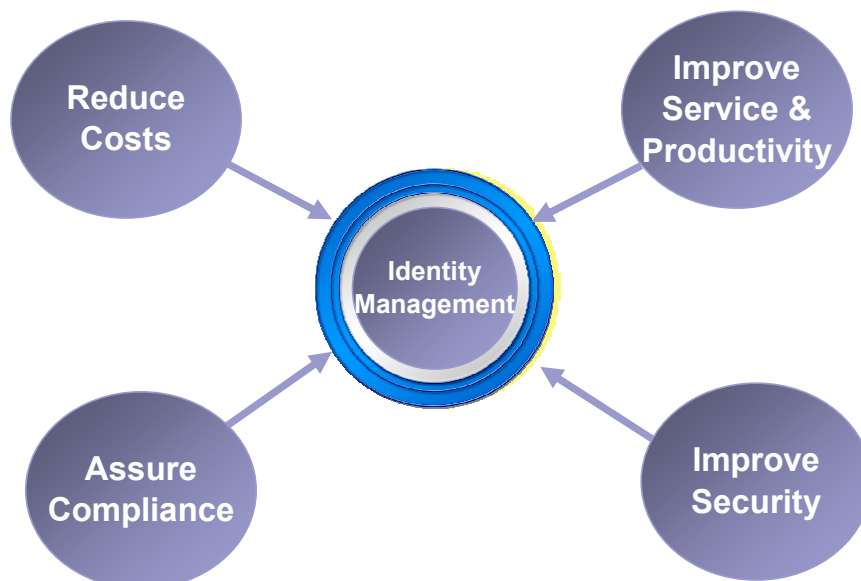


Figure 1 – the benefits of IAM

- **Reduce Costs** – IAM can reduce costs by reducing the administrative effort of maintaining multiple identity databases through an Identity Synchronization Server, automating the account lifecycle process through provisioning and delegated administration tools, and reducing the password management burden with password synchronization and password self-service facilities.
- **Improve Service & Productivity** – IAM can improve service and productivity by reducing the number of usernames and passwords users have to remember through reduced, or single, sign on. Productivity can be enhanced by allowing delegated administration by local administrators, or even the users themselves.
- **Improve Security (Reduce Risk)** – Security can be improved and risks reduced through the proper management of user rights and credentials. By putting in place password management tools, stricter password policies can be enforced without increasing the burden on the helpdesk (or increasing the likelihood of users writing down their passwords to remember them). Most importantly, automated provisioning and deprovisioning ensures rights are allocated in accordance with rules and procedures, and ensures accounts are disabled promptly when users leave.
- **Assure Compliance** – Many organizations have to comply with regulations such as Sarbanes-Oxley or the Data Protection Act and the penalties for non-compliance can be very severe. IAM can assist in compliance by ensuring that user identity data is managed in accordance with applicable privacy or data protection rules, and by using role or rule-based provisioning and access control, ensure that people have the correct entitlements. Workflow tools ensure that approval processes can be automated and auditing tools ensure that records are kept of all changes to entitlements as they occur.

The last point is especially important. If you are working in an organization that needs to comply with SOX or similar regulations, your executives could face jail if they don't comply!

Type 2 – Runaway Train

This is where your project seems to have developed an unstoppable momentum. The list of requirements seems to be growing faster than the project can deliver them and your “to do” list gets bigger while your “done” list stays the same. You need to apply the brakes before you run out of track. The causes of this problem are usually a failure to prioritize and a failure of methodology.

Let's address the methodology first. Whilst iterative development has been adopted successfully by software developers for many years now, its benefits still don't seem to be understood by the wider project management population. An iterative approach means simply many small projects iterating rapidly through the project cycles, rather than a single “big-bang” that tries to deliver everything in one go. The beauty of the iterative approach is that smaller projects are simpler and easier to manage, therefore they are more likely to go in on time. This success breeds success and it's easy to show progress to the project board. Make sure you “time-box” your resources too – set a date to deliver and hit it, drop features or systems if necessary, but deliver *something* on time.

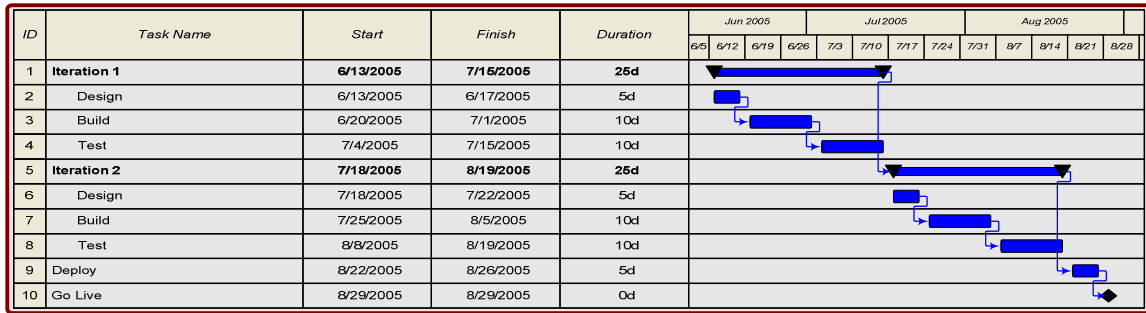


Figure 2 – an iterative approach

If we are going to iterate, we should do so soon, and often. But in order to successfully adopt this approach, we need to have a clear understanding of our priorities in order to know what features should be delivered in what iteration. The best way to understand this is to hold a requirements workshop prior to project initiation attended by executive sponsors (for the beginning and end at least), business process and database owners, owners of application development teams, technical staff expected to be directly involved in the project and, critically, those expected to be involved in the ongoing operation and support of the eventual system. A workshop increases the knowledge of what is proposed, and what is possible amongst the wider stakeholder community, elicits and shares information to all concerned, achieves a common understanding of the project aims and objectives, agrees key roles and responsibilities (both for the project lifecycle and, vitally, for support and management of the operational system). Key deliverables from the workshop are a prioritized list of requirements, a high level solution design and an outline project plan.

Some techniques we find useful for the workshop in building the list of requirements include:

1. Make a list of all identity-dependent systems, including some key metrics and details e.g.

Name	Description	Identities	Details
Active Directory	NOS, email	10,000 staff, 1,000 contractors	EmployeeID as UID
HR	SAP HR and payroll	10,000 staff	EmployeeID as UID
Retail Application	VMS and Oracle	5,000 retail staff	generates own UID

2. Use yellow sticky notes to brainstorm common identity-related problems around the table. Group problems together to identify commonality

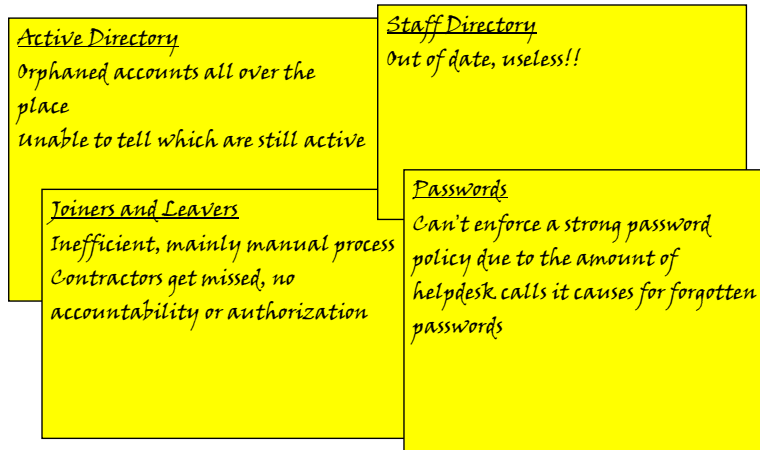


Figure 3 – Brainstorm Identity Problems

- Once you've established the main problems, the next step is to try to assign some metrics to them to determine how much of a problem they really are, as opposed to a mere inconvenience. These will necessarily be fairly rough at this stage, but it's still a valid exercise – just make sure you refine the costs to confirm your assumptions later.

Problem	Cost – how much is this costing now?	Risk – is this a security risk? If so, how severe?	Efficiency – is this an efficiency issue? How long does it take?
New account creation	½ day of a CSA	CSA uses existing account to determine access rights, no approvals process – medium/high risk, doesn't conform to SOX requirements	Can sometimes take up to a week, depending on helpdesk backlog of work
Orphan accounts in AD	unknown – possible licensing implications	High – some accounts may still have access rights	
Password management	High – helpdesk staff spend 40% of their time dealing with password resets alone	High – users are writing down strong passwords	High – takes far too much of everyone's time!

- Now that you have identified your main problems, it's time to turn them into high-level requirements. Remember though to try and keep design out of your requirements – by all means specify a potential solution to the problem, but try to resist the temptation to get too technical at this stage as to how you will fix it. The reasons for this will come clear later, but think about this for a moment – when you go to the doctor's with a stomach ache, do you demand that he removes your spleen, or do you wait for an expert diagnosis?

Requirement	Possible solution	Priority
Password Management	MIIS for password synch, self-service web interface for forgotten passwords	High
Joiners, Movers and Leavers	Provisioning system – perhaps initially only HR to AD, then including other systems and workflow	High
Delegated admin	Web pages for self-service and delegated admin	Medium

Type 3 – Off The Rails

The last type of trainwreck is when everything goes horribly wrong and your project goes completely off the rails. Accounts are being provisioned in the wrong places, changes aren't being replicated, and 1,000 live accounts have just been deleted from

Active Directory. On the one hand the project board are shouting at you expecting more systems to be connected, whilst on the other the operations team are shouting at you and demanding you clear up the existing mess before making things any worse. This trainwreck has two typical causes – one of methodology and one of architecture. Let's address the architectural cause first.

I don't imagine many people would consider building a house on quicksand, so it always amazes me how many people build their IAM architecture on equally flimsy foundations (and how many IAM vendors try to sell people this very approach). The key to a successful IAM project, in our experience, is to put down solid foundations before building too many fancy IAM applications on top. These foundations are provided a **reliable identity data** layer. This diagram shows the idea:

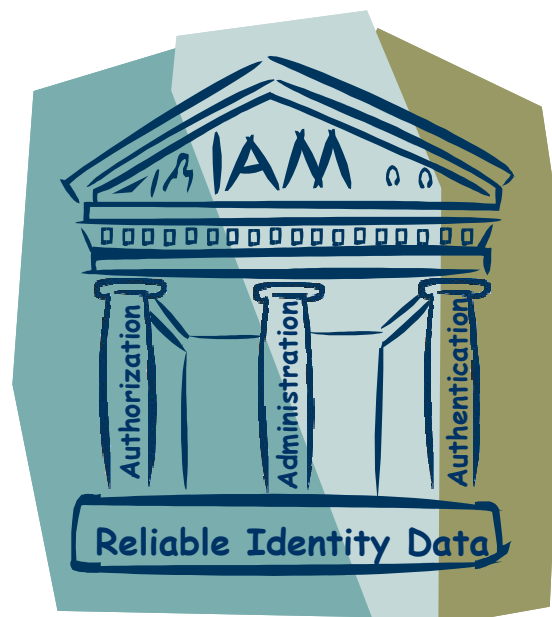


Figure 4 – Reliable Identity Data is the Foundation of IAM

There was a time, way back in the mid-90s, when the current thinking was that all identity data should be contained in a single directory – the enterprise directory. However, no-one told the application developers who merrily continued to develop their applications tightly-coupled to a local database or directory. This means that today's preferred architecture is often a hybrid – a central enterprise directory for LDAP-enabled applications, but also many application databases with their own local users. Therefore we need to try and manage this reality – we need to join all these identity-aware datastores together into a single, managed, reliable identity data layer, possibly (but not always) augmented by a dedicated enterprise directory service. The way we do this is by using an identity integration server (sometimes called a metadirectory). The following diagram shows this is practice.

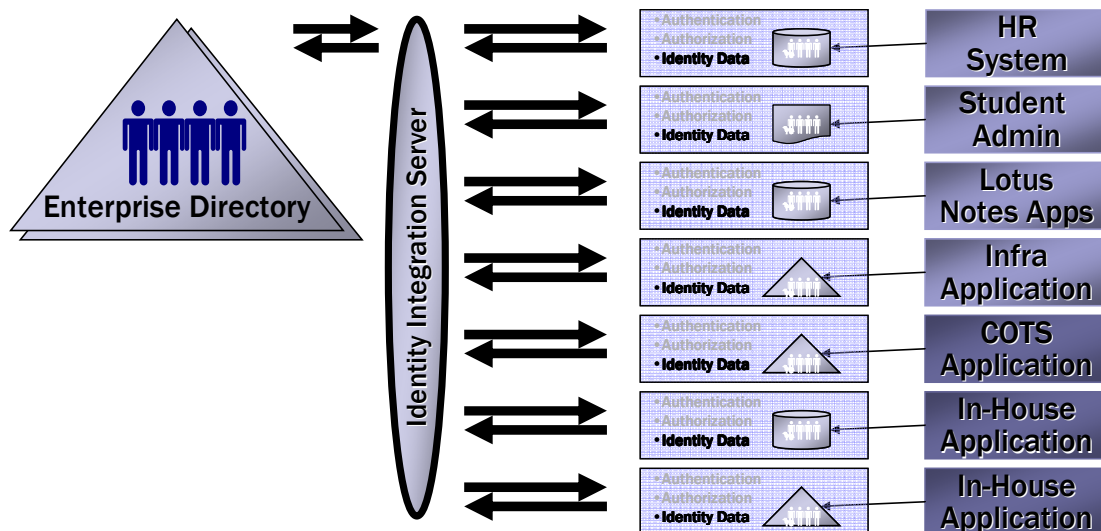


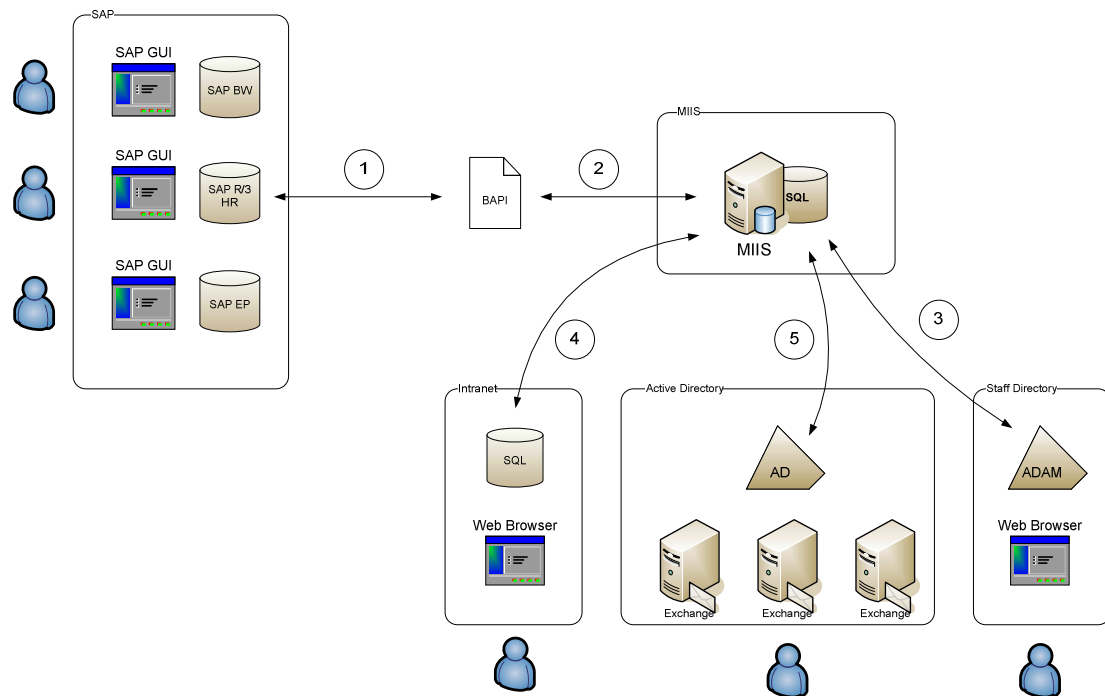
Figure 5 – Identity Integration Server Makes a Reliable Identity Data Layer

The second cause of IAM project going off the rails is a problem with methodology. Many people seem to approach IAM projects in a rather cavalier fashion, often diving in without much design and configuring or developing code “on the fly” – often on production servers. Whether this is because many people implementing IAM come from an infrastructure background, as opposed to software development, or because IAM vendors sometimes make exaggerated promises as to how simple their products are to deploy, I don’t know, Whatever the reasons, it’s a very bad approach and leads inevitably to trouble. The answer, fortunately, is simple: adopt a software development methodology mindset before starting your implementation. The following tips will save you much pain if you use them:

- **3 Tier Environment** – don’t develop on the live servers! Develop a prototype in a virtual environment (using copies of the live data), port the prototype to a dedicated test platform and test it thoroughly before porting eventually to live. During testing, raise bugs formally, fix them in the development environment and then re-test until all bugs are either cleared or manageable.
- **Formalize Change Control** - once you have a working operational system, under no circumstances allow uncontrolled changes to take place on it. Ensure all changes are requested, analyzed, designed, developed, tested and deployed under a formal change control regime. Consider the use of a change control “bug-tracking” or change request tool to aid this.
- **Staged Releases** – think of your system as a software application and release new functionality or bug fixes in staged releases. Make these a point in time and collect together all the new features and fixes and release together – once the development and test cycle has been completed.

The above is second nature to most application developers, but isn’t always as strictly adhered to in an infrastructure context. The other tip I’ve gleaned from software developers that can aid you in your IAM project is to **think functionally** when doing the design. Many IAM people get very excited by the thought of mapping lots of attributes together across systems (whether any application actually needs them or not) and like to dive in headfirst. The resulting mess of attribute flows ends up looking like spaghetti junction. To avoid this you **must** take a higher-level view of the system as a whole. Take your requirements and put together a set of high-level use cases of how the system will work before worrying about attribute flows. Concentrate on how user objects (and there may

be many types of user – employee, customer, contractor etc) will enter and leave the system, map the object flows onto some neat server-style diagrams and described **how** the system will work. The following diagram is an example of the sort of use-cases we use (and, no UML purists, this isn't a proper UML use case)



Once you have described how the system will work, build yourself a prototype that does what you have just described without getting bogged down in too much design detail. This isn't to say that design isn't required – it most certainly is, but all design assumptions tend to go out the window in IAM projects the first time you actually see the data. The best bet is to build a prototype that does the basics, apply representative data from the systems you want to manage then refine your design and prototype until it meets the requirements. You may need to iterate through this cycle several times until your prototype is ready for testing, but take from the people who have done this many times, it's much better this way than a monolithic detailed design phase that turns out to be useless because the assumptions about the state of the live data were all wrong.

Conclusion

If you haven't yet started your IAM project – don't be put off by the horror stories above. If you work with the right tools, the right people and adopt the right approach, you can have a successful IAM framework in place within weeks – not months or years. If you've already started your project and it's looking as if it's about to become a trainwreck, don't despair! Just stop now, review your progress so far and try to find which of the problems above applies to you. If you find it, apply the fixes I recommend. If all else fails, give us a call!