Open Source Game Engines: Disruptive Technologies in Training and Education

Rudolph P. Darken, Perry McDowell
MOVES Institute, Naval Postgraduate School
Monterey, California
{darken, mcdowell} @nps.edu

Curtiss Murphy
BMH Associates, Unc.
Norfolk, VA
murphy@bmh.com

ABSTRACT

While visual simulation technology has changed dramatically over the past 15 years – faster graphics, computing, networking – the business model behind our industry has remained relatively unchanged. The model that developed when a reasonable computer cost $500,000 is still around when we could be using game consoles at $250 each. There must be a better way. The future of defense training and education is very different from what it is today. The trend will be towards small, low-cost simulators that are readily accessible to trainees because more of them are available. They will be available in distance learning modes as part of curricula and as stand-alone applications. When we look at initiatives in the Navy to transform training and education, we see that there will be literally hundreds of training applications developed for very specific requirements. The current business model is not conducive to this end. The way software is licensed, sold, and supported is based on a "few, expensive" model rather than a "many, inexpensive" model. We describe why open source software is a disruptive technology to the defense training and simulation industry and we outline business models that fit this new paradigm in simulation development. We determined that most of what is currently available in proprietary game engines and visual simulation tools is already available in open source. The problem is that the needed functionality is scattered across many disparate open source projects. The framework we describe is a unifying layer that pulls these individual open source products together into a common API that is also open. The advantages to the customer are numerous and include (1) the availability of all source code for developed applications – maximizing flexibility in future enhancements, (2) utilizing free software and the ability to benefit from a community of developers to keep up with rapidly changing hardware and software, and (3) leveraging a large community of developers for support. We conclude with recommended actions our industry should take to benefit from open source software.

ABOUT THE AUTHORS

RUDOLPH DARKEN is the Director of the Institute for Modeling, Virtual Environments, and Simulation (MOVES) and an Associate Professor of Computer Science at the Naval Postgraduate School in Monterey, California. He is the Chair of the MOVES Curriculum Committee and is also the Associate Director for Research for the Center for Homeland Defense and Security. His personal research has been primarily focused on human factors and training using virtual environments and computer gaming media with emphasis on navigation and wayfinding in large-scale virtual worlds. He is a Senior Editor of PRESENCE Journal, the MIT Press journal of teleoperators and virtual environments. He received his B.S. in Computer Science Engineering from the University of Illinois at Chicago in 1990 and his M.S. and D.Sc. degrees in Computer Science from The George Washington University in 1993 and 1995, respectively.

PERRY MCDOWELL is a former Naval Nuclear Power Surface Warfare Officer. He has been on the faculty of the Naval Postgraduate School since 2000, where he teaches computer science, does research in virtual environments and training for the Modeling, Virtual Environments, and Simulations (MOVES) Institute, and serves as Executive Director for the Delta3D open source game engine. He is currently conducting research for his PhD. He graduated with a B.S. in Naval Architecture from the U.S. Naval Academy in 1988, and an M.S. in Computer Science (with honors) from the Naval Postgraduate School in 1995.

CURTISS MURPHY is the project manager for ONR Game and the Navy's open source game engine project, Delta3D (http://www.delta3d.org). He has been developing and managing software projects for 13 years and is currently a Project Engineer at BMH, Associates, Inc in Norfolk, VA. He holds a BS in Computer Science from Virginia Polytechnic University.

Open Source Game Engines: Disruptive Technologies in Training and Education

Rudolph P. Darken, Perry McDowell
MOVES Institute, Naval Postgraduate School
Monterey, California
{darken, mcdowell} @nps.edu

Curtiss Murphy
BMH Associates, Unc.
Norfolk, VA
murphy@bmh.com

BUSINESS AS USUAL

The Navy's *Revolution in Training* initiative is a prime example of how the demand for visual simulation has changed over the years. The vision is to have "on demand" training that is tailored to the specific needs of the Sailor. A Sailor who desires a specific job will have a training program designed based on the individual's background, to include prior experience and training record, as compared to the requirements for that job. A core component of this strategy is the mapping of mission essential task lists (METL) to reusable learning objects (RLOs). RLOs utilize many varied types of media but there is a clear demand for simulation to be one of these.

These "simulators" are far different from what the visual simulation community has been delivering to date. If we look back as recently as ten years ago, simulation did not play the enormous role in training that it plays today. Simulators were typically large "full mission" types of systems that required large, expensive, specialized graphics hardware. The initial business model in the visual simulation field involved hardware integration (e.g. motion platforms, graphics hardware, vehicle mock-ups) and proprietary software development. There were few standards. In fact, when Silicon Graphics made their Iris Graphics Library (GL) open in 1992, renamed OpenGL™, vendors of higher level tools, such as Vega™ from Paradigm Simulation, Inc. quickly found a following. OpenGL was widely adopted but developers typically felt that OpenGL and even SGI's Performer™ were too low level for their needs. This opened an opportunity for the development of proprietary tools built on top of open standards. What resulted was a business model in the visual simulation industry based on proprietary technologies where vendors profited from not only the delivery of the initial product, but also upgrades and improvements, maintenance, and new content. The proprietary nature of this model meant that the customer was often tied to the vendor for the lifetime of the product (vendor lock-in).

When simulators were relatively few and their costs high, this business model made sense. Vendors need to make a profit and this way, they retain a revenue stream well beyond initial product delivery. Customers were also satisfied because they needed to know that the vendor would support the product well beyond initial delivery and also that they had a place to go to obtain needed improvements as requirements and technologies changed.

The following is a summary of the primary business models used in the visual simulation industry.

- **Turnkey systems**. These are usually proprietary, all-in-one solutions. The customer supplies requirements, and the vendor supplies the simulator. The customer pays for the simulator and all upgrades and maintenance throughout the life of the system.
- **Run-time tools**. These are also usually proprietary, but the customer buys a tool with which to build a system, rather than the completed system itself. While the customer now has the power to make changes without going back to the vendor, it is often cost prohibitive to change vendors. In addition to the cost of the tools, there is an investment of engineering time in legacy source code and content.
- **Content creation**. This model can be proprietary, but often involves some standard or de facto standard format. The vendor supplies content only, the customer supplies the application and run-time environment in which to place the content. Changes are only possible if either a conversion mechanism is available, or the customer has the correct tools to edit the content.
- **Content creation tools**. These can also be proprietary but often the file format is the key ingredient. Many formats have become de facto standards such as OpenFlight™, TerraPage™, and 3DS™.

All of these evolved out of an environment where hardware costs were extremely high and simulation had not yet earned its place in defense training and education as it has today. We do not mean to imply that these business models are no longer valid. In fact they are not only valid but they are valid for the same

reasons for which they were developed. But the role of simulation in defense training and education has clearly changed. There are requirements now that were not imagined ten years ago.

Which of these business models fits these new requirements? We believe that none of them fit well and that we must think differently about how we build and maintain simulation systems to adapt to these new requirements. Clearly, a business model that was designed for the acquisition of a five million dollar training system is not likely to be appropriate for a twenty thousand dollar simulation object used in the Navy's Revolution in Training. But what is appropriate and how does it apply to today's requirements?

IS COTS THE ANSWER?

For the past several years, we have seen a drastic shift towards commercial off-the-shelf (COTS) hardware and software solutions. Is this the solution to the simulation acquisition problem? What makes COTS so attractive? There are several reasons why COTS is viewed as a winning approach:

Point 1: *Low (to negligible) engineering and development costs.* By sharing the engineering costs with the entire market, you save money.

Counterpoint 1: This only works if no adaptation is needed. If what you need is exactly what the product offers, then this model works. Otherwise, the costs to adapt COTS to what the requirements really are can often be more expensive than building from scratch. Microsoft Office™ is a good example of where COTS works well. What little adaptation is needed is provided in the form of macros or preferences.

Point 2: *Swappable "best of breed" approach to system development.* By buying COTS systems, you can define an architecture that allows you to swap out one COTS product for another if a better or cheaper product comes along after initial system development.

Counterpoint 2: This only works if the architecture for the objective system is modular. This is the desirable way to design systems but unfortunately, this is not achieved as often as we would like, at least not to the extent that modules are "swappable".

Point 3: *Reusability of modules.* By purchasing COTS products, they can be paid for once and reused on multiple simulation projects thus amortizing the customer's costs.

Counterpoint 3: Again, this only works for a modular, open architecture. How many defense simulations that were acquired in the past year were of the turnkey variety? This remains a dominant model for defense simulation acquisition (Nash, 2000).

Point 4: *Low maintenance costs.* By using a COTS product, you share maintenance costs across the entire market, thus saving money. This includes both upgrades, help assistance, and product training.

Counterpoint 4: This works well for the turnkey and content creation tool models described earlier in that the maintenance we are speaking of is on the product that is purchased. When buying a COTS development tool (like a run-time environment), this savings only applies to the tool itself, not what is built with the tool – and this is what we really need maintenance on.

COTS appears to be a clear winning solution if and only if the product is a commodity. If the vendor of a COTS product knows that his product has reached commodity status, then by definition, there is no obvious difference between his product and his competitor's. The customer will switch from one vendor to another at will as prices fluctuate, service improves or declines, etc. This leads the vendor to a more open model. Obviously, being proprietary in a commodity market is a losing proposition. How long would a company manufacturing light bulbs with a proprietary connector stay in business? This implies that the customer retains flexibility in choice. The customer can pick and choose who to do business with based on previous performance, cost, responsiveness, subject domain expertise, etc. The choice is not made on the basis of "Our current system was built by them so we have to go back to them for improvements." There is no lock-in.

On the contrary, when we choose a COTS solution in a market space that has not yet reached commodity status, there is great potential for lock-in. In the early days of visual simulation, with few to no standards and little competition, a vendor could develop a proprietary API in which to deliver a training system. The customer accepted vendor lock-in because they had no real choice. That was when the visual simulation community was young. The question now is – *Has the visual simulation industry reached the point of commoditization?* If it has, or if components of the industry have, then we need to rethink our business models.

The focus of this paper is on the *visual* part of visual simulation. Specifically, we contend that real-time computer graphics are a commodity and that open

source computer game technology is a disruptive technology to our industry. We will first define what a disruptive technology is and will follow with data to support our claim. We conclude with actions we can take as an industry to embrace this change and to benefit from it, not only strengthening our industry but also producing better simulation systems for our customers and growing our customer base throughout the worldwide defense industry.

DISRUPTIVE TECHNOLOGIES

Not all change is disruptive. In every industry, changes occur over time, all of which improve some aspect of that industry. Engineers see the world as a place full of things that could be made better. Most of these improvements are what are called *sustaining technologies*. In these cases, the product or industry improves by some amount, but fundamentally remains the same in the way it operates, the measures of success, and the customer base.

Christiansen (2003) contends that real innovation happens as a result of disruption. But disruptive change is hard to identify and even harder to react to. Disruptive change does not look like sustaining change. In fact, it often looks like the opposite – like a bad idea, not a good one. Disruptive technologies:

- tend to apply most to an underserved (niche) market,
- are of lesser quality than the mainstream technology, but
- appeal to users on a different set of performance characteristics than their mainstream competition, and
- are *never* asked for by the customer.

An Example

We'll use Christiansen's example of disk drives to explain this further. Initially, during the mainframe era, storage capacity was the only metric of performance in the disk drive market. As long as storage capacity increased every year (sustaining technology), disk drive manufacturers had a market. Then the personal computer started to take hold, and the rules changed. The sale of disk drives was initially a niche market. How many disk drives could be sold to PC owners vice mainframe owners? How much could a PC owner pay for a disk drive as compared to a mainframe owner?

Compare this to today's simulation market. The Navy is willing to bear a significant cost for a software upgrade to a simulator that has a full motion base and projection displays? But how much is the Navy willing to pay for a software upgrade to a lube oil strainer simulator that involves eight moving parts?

Returning to the disk drive example, a drive made for a PC is clearly inferior to one made for a mainframe. It has far less capacity. But what does it have that the mainframe drive does not? It is small. It wins on a different performance characteristic than the mainstream competitor.

As the story of the disk drive industry unfolds, we see that all the major players in disk drive manufacturing in the mainframe era died when the PC era took hold. Why? They were unable to see the disruptive change that was created by the Winchester disk drive. Why didn't they see it? How many of their customers asked for a 5¼" drive? None of them, because they all sold exclusively to mainframe owners. By the time the writing was on the wall it was too late. The 5¼" drive market was already established and it was clear that the PC market was going to be enormous.

The good news here is that disruptive technologies tend to be hyper sensitive to first movers. This has powerful implications for our industry. If we identify and embrace a disruptive technology early, we can greatly influence its development. As early adopters, we can direct the technology to meet our goals and objectives. The bad news is that it is hard to identify disruptive technologies and even if we do, early adoption is often at a performance level that is below acceptable use standards. In the disk drive example, the very first 5¼" drives had so little capacity that they were hardly useful at all. But as the co-evolution of the disk drive market and the PC market developed, this changed rapidly.

Are Computer Game Technologies Disruptive?

To determine if our hypothesis is correct, let's compare game technologies to the list of criteria for disruption to see if indeed game technologies are disruptive to the simulation industry.

1. Does it appeal to an underserved market?

   Yes. It appeals most directly to the warfighter. Game technologies make it cost efficient to train the warfighter directly with a technology they are already comfortable with. Task Force Excel specifically targets the warfighter for this type of training.

2. Is it of lesser quality than its competition?

Yes. We do not foresee running TOPSCENE* on an Xbox™ or a standard PC in the near future. Many applications currently in use require specialized graphics hardware. Whereas, game technologies specifically target lower performance commodity level hardware.

3. Does it appeal to customers on a different metric?

   Yes. These customers are willing to give up some graphics performance in favor of increased mobility, durability, ease of use, and flexibility.

4. Is the target customer asking for it?

   No, but this is changing rapidly. While it is certainly true that we are seeing many more game-based training applications each year, it would be hard to argue that a typical Soldier, Sailor, Airman, or Marine envisions using games for training. Even in cases where they are, often program management does not agree to this approach. We anticipate than in time, not only will warfighter and program managers be asking for this, but games will be an essential every day part of training.

The key to identifying a disruptive technology is to realize that the "rules" have changed in an industry. The rules of an industry would define what is valued and what is not and how profit is to be gained. This changes over time. If game technologies are disruptive to the simulation industry, what performance characteristics do game technologies win on? There are two points of view to consider: the acquisition perspective, and the end user perspective.

From the acquisition perspective, when cost and flexibility are critical, what the customer wants is reusability on a grand scale and avoidance of any form of lock-in that could limit future acquisition choices. This argues for open source. But does it have to be game technology? The end user's perspective will answer this question.

From the end user perspective, visual fidelity used to be the driving attribute. We cared about polygon counts, environmental effects, networking, and interoperability. We could only get these things on specialized hardware using proprietary software, so open source was not an option. In the meantime, the computer gaming market was building better and better game engines that ran on commodity level hardware. Today, we can get almost everything we want on a standard PC graphics card. The rules have changed. The end user cares about mobility, deployability, ruggedness, ease of use, and size. In addition, they "want it to look and feel like a video game", meaning that they want artists making their environments. They want well designed interactions that they are able to use "out of the box". A typical Xbox™ or Playstation™ multiplayer game is being played by the average eight year old ten minutes after it is brought home. This is a design metric that should be applied to training systems. Priorities have shifted away from pure performance towards usability and utility – these are strengths in the computer gaming industry.

In summary, video games are attractive because:

- they run on just about any hardware, but particularly on reasonable cost high performance hardware,
- they are easy enough to use that a child can install and run them without assistance,
- multi-player games are almost a commodity themselves, and
- they look as good or better than anything the defense simulation industry has produced.

Taken a step further, we can add what is attractive about game console technologies (e.g. Xbox™ or Playstation™) to include:

- they are instantly on (no visible boot cycle) and there is no operating system to get in the way or to break so the application won't run,
- they typically use special purpose interface hardware (e.g. game pads, joysticks, etc.) because they recognize that a keyboard/mouse interface isn't a good fit for everything, and
- they're durable, able to withstand abuse from kids over a long period of time.

Figure 1 shows price and performance graphs for the graphics hardware we use in our industry. It suggests that the convergence in performance between game console hardware and PC graphics hardware makes consoles an attractive option.

* Naval Aviation mission rehearsal system. Very high fidelity running on high end graphics computers.

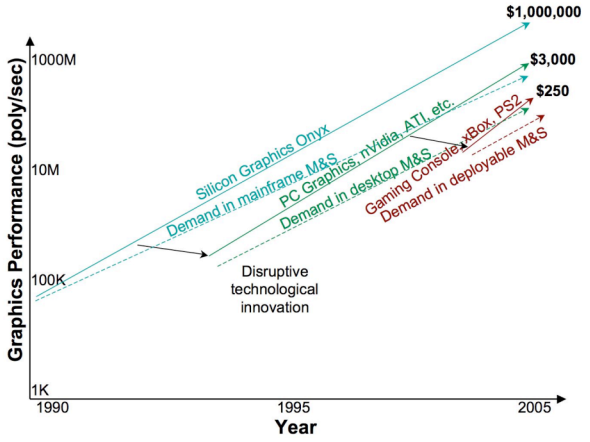

Figure 1. Performance of graphics hardware

It seems obvious that the simulation industry has a place for game technologies. Why haven't we seen a shift in this direction? We see two reasons:

1. The use of game engines and open source software is disruptive to our industry requiring that we change the way we build systems. We have not addressed this change. We continue to apply our old business practices, developed when you had to have an SGI to do anything of value, and an SGI cost hundreds of thousands of dollars, apply to this new world, where an Xbox™ costs $250.

2. The game console market is closed to us. Both Sony and Microsoft have a business model for the entertainment industry that is based on selling game titles, not selling game consoles. Both companies sell hardware at a loss in order to make a profit on the games themselves. If they sold consoles to the military for training, where would their profit come from? They are not interested. It appears to be nearly impossible to effectively use this market to meet military training requirements.

Notable exceptions to this are Full Spectrum Warrior™ built for the Army, and First to Fight™ built for the Marine Corps. In this business model, the developer is paid for and delivers a special version for the military but retains the rights to sell a non-military version to the public. This is very innovative and appears to be very successful, but it is not scalable. A game developer will happily take government funding to build a trainer for urban assault if they can sell a version to the public. How do we convince them to build trainers for logistics, air conditioner valve repair on Naval vessels, or even rules of engagement? Most of the training we need to do does not have the necessary appeal to the public at large to make this model work. We need another way.

OPEN SOURCE: A NEW APPROACH

We have developed an open source game engine for the development of Navy training applications (McDowell, et al, 2005). It is not our intention to solve all the Navy's training needs with this engine. As stated earlier, the existing business models still apply. But there is a new class of application (and customer) to which the existing business models do not apply. Smaller applications and applications where flexibility is key will need a new business model to flourish.

Why Open Source?

What does open source software (OSS) do that proprietary software does not? The first, and most obvious response is that it is free. While true, we think this is the lesser of the reasons for embracing open source. The big reason is its openness which retains flexibility in the development process. Open source implies commoditization of the rendering engine. Consider all the available rendering engines on the market today, to include game engines, visual simulation tools, etc. How much overlap is there in functionality between them? Despite the obvious overlap, it is uncommon for an application built in one to move easily to another. In fact our point is that exactly the opposite is true. While almost all rendering engines or image generators have some unique features that the vendor might use to sell the product, the large majority of the feature set is common to all. Then why are they proprietary? In order to get access to that small set of advanced features, flexibility is lost. This is most often not a fair trade. In the entertainment industry, having a feature in your engine that no other engine has can make or break a title or series of titles that use a single game engine. In the defense training and simulation industry, we rarely need this. If we had 80% of what all rendering engines had in open source, we could build a large percentage of applications with it. Those applications that require a special feature may have to go the proprietary route. But most of what we want is available.

As we investigated this further, we realized that not only was it a good idea to commoditize rendering engines through open source, but that most of what we wanted was already available in an open source format! Very little code needed to be written. By unifying OpenSceneGraph (OSG) for rendering, Cal3D for articulated figures, OpenDynamicsEngine (ODE) for physics, OpenAL for audio, etc. we were able to build

an engine consisting of 1.3 million lines of code while writing just 50,000 lines ourselves. That's just 4% of the total! Further, we benefit by leveraging the large existing communities of all the open source projects that make up the engine. We do not know who will write the code to adapt the engine's renderer when the next version of OpenGL is released, but we are certain that the Navy will not be paying for it. It will be provided by the OSG community of developers. The savings in engineering costs alone for maintenance of the engine over its lifetime by using the open source community is staggering.

A common criticism of open source is its lack of support. This is a fair criticism particularly for small OSS projects with a small developer base. There are simply too few developers (who are not being paid to answer questions) to solve every problem you may have. But even in this case, the situation where a proprietary software tool has a bug that needs fixing is resolved. It is extremely frustrating for developers to identify a bug in a tool, report it to the tool vendor, then wait (and hope) that it gets fixed in time to make their deadline. With open source, even if you have to do the debugging yourself, it is possible to do it. Then you become a contributor to the community when you send in your fix thus making the tool better. If everyone does this (and all successful OSS projects pride themselves on developer participation) then imagine the quality of the tool. In fact, it is argued that the primary reason why Linux is more stable and secure than Windows™ is the number of people who are viewing the code (Raymond, 2001).

In terms of direct costs, a production level game engine can cost anywhere from $100,000 to $1M. These will usually have recurring maintenance costs. Game engines are revised often and if a customer is not under maintenance, they usually will not receive updates. Also, when a new version of the engine is released (not just an update but an entirely new version), the customer typically has to pay again, albeit possibly at a reduced cost. Visual simulation engines are typically cheaper, but they often have run-time licenses that limit the ability to deploy an application. Even showing a prototype application to sponsors or colleagues can be frustrating under this type of licensing agreement where you don't have the right to take your application anywhere you choose without an added cost. A cost is incurred for every instance of the application you deliver. This is particularly distasteful to program managers who feel that they already paid for the software, then paid for the development of the application, and now they have to pay a third time to send it where it is needed.

This was a primary motivator in the development of a forward observer trainer we built using our open source engine. FOPCSIM was first developed using a common visual simulation toolkit with a run-time license. We were unable to send it to the people who wanted it because they did not have funds to cover the cost of the run-time license. We re-implemented it in open source and it is now being adopted into the Marine Corps family of tactical decision trainers and is being used in allied countries around the world. This is what we intended when it was built.

In summary, the primary reasons why OSS is attractive to the defense simulation community are:

1. Flexibility. You own the code and therefore have all the flexibility you could want in terms of who develops the application, who maintains it, etc.
2. Cost. Not only is the software free, but you get the benefit of many hours of engineering labor contributed by the OSS community.
3. Support. Usually one of the primary knocks on open source, this is actually a strength *if* you leverage strong projects with a large following. There is no fear in the OpenSceneGraph community that OSG will cease to exist. All those developers are a support asset to anyone who uses OSG.

HOW DO YOU MAKE A PROFIT?

For a business model to be successful, it obviously needs to identify ways in which a profit can be made. This is true here as well. The issue of how open source software in general fits into the government acquisition process is a hotly debated topic of concern (Hahn, 2003). There are no answers at this point but it is clear that open source will have a place in government IT infrastructure for many years to come. Linux and Apache, to name a few, are already entrenched in certain government organizations. We must keep a watchful eye on how other open source applications fare in the government to learn new ways to help vendors to thrive in a changing business environment.

There are several models that clearly work well in the open source environment. These are best described in terms of Figure 2. There are two sides to the development process: development and delivery. On the development side, the programmers are writing source code directly to the engine API and they are writing script (Python in our case) as a higher level abstraction to control behaviors and general content. The artists are using modeling tools such as 3D Studio Max™ and Adobe Photoshop™ to develop visual

content that will be manipulated by the programmers. There is also often a level editor that allows a non-programmer to control content at the highest level.

The users are on the delivery side. They directly use the application that was developed by the development team. They might use the level editor to alter content and they certainly will use a scenario editor that allows them to alter missions or to make small adjustments to scenarios. These are common in modern video games, but fairly uncommon in training systems. All these applications run on the game engine on standard hardware.

By commoditizing the engine (the hardware has already been commoditized by graphics card vendors) we move the business model up a level to the tools.

1. Application development. There will always be a need for new applications. Vendors who can solve real training and analysis problems with high quality applications will not care if the underlying software is open source or not. It is also important to note that applications built on an open source tool are not necessarily open source. Only additions that are made to the open source tool itself must be made open source.

2. Content creation. Similarly, as we abstract the development process away from the engine and up towards the tools, content becomes the critical factor. All types of models and art will always be needed.

3. Tools. The assortment of applications that will need to be built is so varied, there is no way to build a single development tool/editor optimized to build them all. A special tool for flight simulation, urban combat, maintenance training, etc. could be developed with that specific application area in mind.

4. Advanced modules. While the code of the engine is open, we make no claim that the entire engine must always be open. If a developer builds a night vision module that contains proprietary algorithms, this should remain proprietary if they so choose. This would not be a commodity item. Proprietary modules for surf zone modeling, avatars, SAFs, etc. could also be built for the engine.

We make no claim that this is an exhaustive list. Certainly entrepreneurial vendors will discover new ways to work with the defense training and simulation industry using these same technologies.

It is critical that both sides of this business relationship understand each other. Industry needs to understand that the government is obligated to the public to maximize the value of each dollar spent on training systems. Government needs to keep in mind that industry needs to make a reasonable profit so that it can remain a productive, contributing part of the defense simulation industry.

CONCLUSIONS

The landscape in training and simulation is quickly changing. Looking at the annual I/ITSEC exhibition, it seems obvious that games will have an important place in our future. But game technologies have fundamental differences with conventional technologies and those difference give us pause to think about their impact on our industry. Both game technologies and open source software have huge implications to our industry.

We have made a case for a wide variety of types of

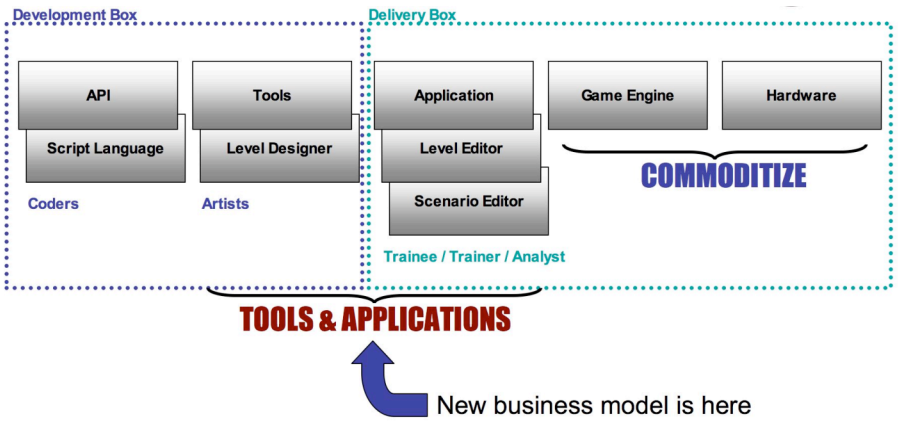

Figure 2. The key components of the development process. Developers are on the left, end users are on the right.

simulation in defense training and education. Some of these demand a different business model to make them cost effective. We described the concept of disruptive technologies and described how open source game technologies and consoles fit the description within our industry. We have described an effort to bring open source software to the visual simulation industry through a new project that leverages many existing open source products. Finally, we have described several ways that industry can leverage open source game technologies in a profitable business model.

In order for the defense training and simulation industry to benefit from disruptive change caused by game technologies, there are several recommendations to consider:

1. Encourage the use of open source where appropriate and actively work with industry to find pricing models and new business models that work for both government and industry. It is better to proactively influence the effect of these technologies rather than have them influence us.

2. Let the community participate! This is what makes open source work. We should not only leverage open source software but we should be active participants in OSS projects. By helping making the OSS products we use better for everyone, we make them better for ourselves.

3. Standardize on commodity level components – don't pay twice for exactly the same thing! This will align both vendors and customers for the fourth recommendation:

4. Content is king. Funds spent on developing content are usually well spent. Funds spent on re-engineering applications may not be. We want to maximize reuse everywhere possible. There are so many applications that need to be built that it is silly to think that a vendor has to maximize profit on a single application when there are many application to follow.

5. Leverage our use of open source software to develop an open hardware platform. Every year there are several failed endeavors to build a Linux-based game console. The defense industry market alone could make one of these architectures successful. It would provide us with a very inexpensive, stable platform for delivering training applications to the warfighter comparable to the Xbox or Platstation platform but open to our use and scalable to fit our requirements.

Game-based simulations will soon be pervasive throughout the defense industry. All involved will seek out the most beneficial business model for their needs. The use of open source software in training and simulation is inevitable. By taking an active role in helping OSS find its place in our industry, we serve government and industry alike towards building the best possible applications for the people who matter most – the warfighter.

ACKNOWLEDGEMENTS

The authors wish to thank the Naval Education and Training Command (NETC) for their support of the Delta3D open source game engine initiative. Other sponsors include the Joint National Training Capability (JNTC), the Navy Modeling and Simulation Office (NMSO), and the National Geospatial-Intelligence Agency (NGA).

REFERENCES

Christiansen, C. M. (2003). *The innovator's dillema*: Harper Business.

Hahn, R. W. (2003). *Government policy toward open source software*: American Enterprise Institute Press.

McDowell, P., Darken, R., Johnson, E., and Sullivan, J., (2005). Military uses of an open source game engine, *Presented at the IMAGE 2005 Conference*, Scottsdale, Arizona, July 2005.

Nash, T. (2000). The Giant Stirs: The U.S. Training and Simulation Industry, *Military Training and Simulation News*, 2(6). pp.24-28.

Raymond, E. S. (2001). *The cathedral and the bazaar: Musings on linux and open source by an accidental revolutionary*. Cambridge: O'Reilly.