APPLYING METRICS DRIVEN DEVELOPMENT TO MMO COSTS AND RISKS

A White Paper by Larry Mellon sponsored by Versant Corporation

255 Shoreline Drive, Suite 450
Redwood City, CA 94065
email: info@versant.com
Main: (650) 232 2400
Fax: (650) 232 2401
web: www.versant.com

Abstract

Our industry is in the upper third of software complexity projects in terms of raw code size plus maintaining and constantly updating 3-D, real-time virtual worlds: we must also retain customers with economies under constant hacker attack and griefers spoiling the user experience. Yet many MMO projects to date have had very few system level tools to analyze the system as a whole, leaving programmers dependent on log files for postmortem debugging, designers dependent on skimming the user boards for trends and management dependent on guesses and antacids to get through the day. Metrics tools have proved their worth in measuring aspects of system performance, project completeness and player behavior in several MMOs. This white paper looks at several useful metrics that have worked across multiple games, how to apply metrics in development and operations and how to transform several individual grassroots metrics into a MMO-standardized set of metrics suitable to measure and compare MMOs and their components.

Introduction

Migrating to online game play and constant content development has been difficult for our industry. Some have failed for lack of scalability or stability, some for content starvation and some, well, we just don't know why. A large part of the puzzle is a historical lack of metrics oriented development in the game industry. MMO game services have added tough new requirements such as stability, scalability with low latency and massive increases in software size, team size and gameplay complexity. MMOs also require code practices capable of affordable content evolution and years of cost efficient operations. These are tough problems requiring constant, measurable attention at the system level, the business, software and operational architectures all while evolving your customer base. If you ignore metrics, at least one of these many new and constantly evolving problems will fry you.

Over the past 20 years of building large-scale parallel and distributed systems, I have constantly found a metrics tool to be invaluable (especially when it replaces raw data with system-level views), and there have been a number of metrics success stories in the MMO space as well. But even these MMO successes have only scratched the surface of what a Metrics Driven Development focus can bring to the full set of development and operational tasks of an online game.

Finally, lack of standards and metrics make it very difficult to evaluate middleware or open source technology for your game. It is time to stop making excuses about how different our games are and how young and immature the MMO industry is. We as an industry are at a very Darwinian point in time: those who cannot stabilize risk and control cost in this new era of online games will be sucking on glaciers to survive.

This white paper discusses paths of spreading today's use of metrics into broader areas in developing and operating MMOs:

Metrics Driven Development (MDD) is not a process per se, but rather finding a way to quantify your tasks and risks to best focus your team's limited resources on your project biggest problems.

MDD increases team efficiency by increasing individual developer efficiency: metrics are a valuable tool, providing a faster way to find bugs or measure potential solutions. When you measure -- and publish -- something critical, say, the number of clients a server can support one year from launch, that task becomes actionable. Developers and producers know what they are working towards; and management can track historical and current progress against long-range milestones. Imagine knowing six months in advance of your Beta release that most features are ahead, but scalability is lagging. Simply shifting a couple of programmers over to scale related tasks gives you an extra man year of labor against the critical launch risk -- problem solved. What is the typical MMO process now? We find out half way through Beta testing that the servers aren't scaling, pull out the sleeping bags and everybody lives on the edge of disaster until launch…

Metrics Driven Development provides a potential escape path from our current development cycle of guess/change/hope (repeat ad infinitum) to a measure/change/measure cycle.

Which would you rather use? Which do you think will help your business survive?

MMO risks

For every large project I've ever been on, the project schedule usually started with a metrics system that was in the "*nice to have early, but we need features first*" pile, and it has always turned out that metrics should have been in the "*do it early or watch your project suffer and slowly die*" pile. This effect usually occurs when a project grows enough to start tripping over the *Law of Large Numbers*, (see sidebar).

The Law of Large Numbers holds true from your business architecture to your software architecture to your pipeline implementation: for example, scalability defects are usually very deep in the roots of your code and processes, yet usually aren't found until the project is almost complete and running at scale, leaving a nasty surprise for your schedule.

Law of Large Numbers

Any small number, when multiplied by a sufficiently large number, becomes a large problem.

The number of connected clients, simulation event overhead or the number of CS calls per day: any expense in your MMO game service should be measurable.

If you cannot estimate risks such as scalability before construction and you cannot measure scalability while building it, you will not have a scalable system at the end. You'll have a constant, panic driven system, kept afloat by a steady stream of programmer's blood.

Performance and Scalability risks

An interesting example is a Data Agent (DA), from a fielded MMO. Sending the entire dataList, every 30 seconds for each live player generated a harmless overhead in small-scale testing, but jumped to over 3 GB per minute under full-scale testing. With architectural bandwidth budgets, you can usually catch these defects at the review stage, before any unnecessary code is built. If you don't, you can waste several months building the system, then spend several weeks frantically measuring and reducing all bandwidth costs, stalling other work until the final DA cost is known. This difficult, nondeterministic bug was easily caught during large-scale load testing with metrics tools, which measured how much bandwidth each Data Agent consumed, where the data went, and measured when the programmers had successfully brought it down to the target bandwidth metric. By the end of that MMO development project, metrics were so effective that most of the server team was tasked by metrics output from large-scale, daily load tests: *hard numbers had made tasks tangible for the programmer and actionable for the management team*.

Iteration Rate in Development and Operations

Like scalability, Iteration Rate – the time it takes to rebuild the system for play testing / game play - is fast and painless in the early stages of a project but becomes a bottleneck later on that could determine the success or failure of the entire project. Many projects start off with a "temporary" pipeline that takes only a few hours of engineering time to jumpstart the critical feature prototyping stage. But as the *Code Mass* and *Team Size* increase, the pipeline and the game become increasingly brittle to work with. Remember, *the game is a tool used in building the game*. Content Production and Game Operations are both severely handicapped when it takes several slow, error prone build/test/deploy cycles to debug and restart live servers.

The iteration rate problem is magnified in operations, with angry users and terrified executives screaming for service to be restored (note the nationwide outrage when Blackberry e-mail was down for three hours!).

There is a *Creative Risk* involved as well. If you have a slow iteration rate, you can't quickly and reliably try ideas from the design team on a stable server, where it can be play tested and measured, feeding key data back to the design team, where they can remove problems and polish gameplay.

Rapid iteration requires an investment in automation and changes to code and processes, but a faster, stable and scalable production cycle is invaluable over the lifespan of an MMO.

Player Behavior

On the business side, visibility into player behavior -- in the game and in their community -- and operational metrics allows you to tune your money funnel from day one to year ten. Automated metrics can extract and aggregate the data affecting your top-level business metrics: cost of customer acquisition; cost of customer service; cost of customer retention.

Given the obvious value of metrics, why are they not in broader use? It is far too common for our industry to spend as little as possible on anything that's not a game feature. Tools in particular take a big hit here, and especially system tools that require infrastructure to collect, aggregate and analyze system level data and defects. While cheap tools worked with smaller, deterministic games, under-investing in support software for a 10 year plus service, failing to deal with scalability or not ensuring a rapid build/test/deploy cycle before launch are all potential death blows.

Define MMO Metrics Categories

It is well past the time where our industry should standardize common functionality, using common metrics to allow meaningful comparison of the ever-growing set of middleware components, open source modules and entire game engines. Validating the standards is possible by tests against an open source MMO engine, such as World Forge, or participation from middleware vendors, academia, and various serious game groups, such as real-time military training or distributed virtual world classrooms. The following is a proposed "starter set" of metric categories, commonly useful metrics and how to collect and apply them. Game metrics group into the "3 P model", with common and unique data instances per MMO:

Player behavior, engine **P**erformance, and **P**ipeline throughput

Definition and Applications of Metrics in MMO Games

Metrics are the mission-critical parameters by which you will evaluate your game's true current state and progress over time. A MMO metrics system can easily collect half a terabyte of data per day or more. But without automation support to collect, aggregate and analyze these parameters to find system-level patterns, your entire team is left swimming in a vast sea of raw data -- unable to find the problems they are looking for in time to be of value.

Remember, garbage in, garbage out! When you are using metrics to make critical business or technical decisions, you need the data to be accurate. Calibrate your metrics system for variance and track some key metrics with multiple views, like making sure messages sent and messages received from all processors add up correctly (you'll be surprised how often they do not). And remember, a metrics system is just like every other chunk of software: there will be bugs to hunt down there too.

Metrics on Metrics are extremely valuable. Placing metrics on your metrics system gives you invaluable data on what types of metrics people look most for, what metrics are you collecting that are not being used and what bottlenecks prevent rapid access to critical data.

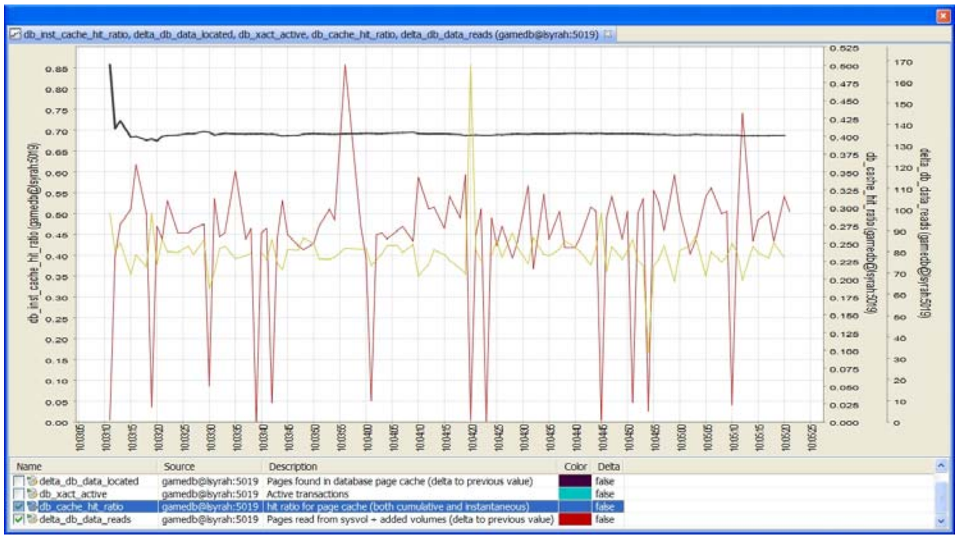
Example: Application Stability Metrics

One of the first metrics we tried on an MMO was a client uptime measure. Even though we showed the client could not stay alive for more than an hour, making things more stable was put off until close to launch. This appears to be due to the fact that game development was not considered to be slowed by the lack of stability, as most play testing sessions were 20 to 30 minutes. Anytime the client began crashing sooner than that, it became a high priority, but only because it slowed the game development!

Example: Performance / Scalability Metrics

Client-side latency is a key metric to know if customers are satisfied, but to improve latency you need to measure and improve several different server-side metrics. Processing times per transaction inside each server simulation, the frequency and cost of inter-simulation accesses and simulation accesses to the database are critical scalability and latency factors to control. A classic client-side "ping and respond" message gives you the round trip data you need, but *validity requires many samples from many clients* and a time driven view of min/max/average/frequency latency values to capture short, ignorable spikes of poor lag versus long, unacceptable periods of poor lag. More valuable system metrics are: connected clients per server, average and individual cost for each client/server transaction, where cost is measured in both time of service per transaction, average/min/max/total bandwidth, and cost/frequency of each packet type.

Finally, note that many an elegant scalability test has been spoiled by poor test procedures: databases with small amounts of data; too few test servers slowing down total server events per minute (a metric for test and live); unrealistic numbers of gateway or simulator processes and many other problems can give you a highly unrealistic view of how well your system scales.


db_inst_cache_hit_ratio, delta_db_data_located, db_xact_active, db_cache_hit_ratio, delta_db_data_reads (gamedb@lsyrah:5019)
db_inst_cache_hit_ratio (gamedb@lsyrah:5019)
db_cache_hit_ratio (gamedb@lsyrah:5019)
delta_db_data_reads (gamedb@lsyrah:5019)

Name	Source	Description	Color	Delta
delta_db_data_located	gamedb@lsyrah:5019	Pages found in database page cache (delta to previous value)	🟪	false
db_xact_active	gamedb@lsyrah:5019	Active transactions	🟦	false
db_cache_hit_ratio	gamedb@lsyrah:5019	hit ratio for page cache (both cumulative and instantaneous)	🟦	false
delta_db_data_reads	gamedb@lsyrah:5019	Pages read from sysvol + added volumes (delta to previous value)	🟥	false

Figure 1: Database Metrics Tool Example

Automated Monitors help operators understand live systems for debugging and optimization, and when changes across software releases show an increase or decrease, per module. For example, Versant Witness provides stats on database disk reads and cache hits.

Example: Pipeline Speed and Stability Metrics

Pipeline metrics are surprisingly easy to capture. Many automated build and test systems collect this information but programmers and managers tend not to use it, probably because builds are generally fast and hassle free at the start of a project. With no visibility into the build times and build failure rates, they are ignored until they are clearly stopping the project from progressing. A common phrase heard on painful projects: "the build time is probably not that bad, and it probably doesn't fail often enough to be worth fixing. After all, it's not a feature, and we're only a year from launch…" But when the Development Director sees the true cost of a poor pipeline, when multiplied by the amount of total team time lost per bad build, improving the pipeline becomes a priority! Common pipeline metrics are overall iteration rate (from build start to deployed test clients running against deployed test servers, and copy time of valid clients back to the development team) and execution time/failure rate per pipeline stage.

Software stability is a fascinating pipeline metric. Using data from your source code control system and your bug tracking system, you can easily find out which files are the most frequently involved in a "claim fixed" check in. When trying this on a previous MMO, we were shocked that five files had been involved in a truly massive number of bugs. One file turned out simply to be a common includes and constants file, but the other four turned out to have client/server protocols with massive timing and data dependencies that could be accidentally broken by almost anyone working almost anywhere in the code (explaining our horrendous recurring bug ratio and go back costs). We had to completely re-factor those four files before we could stabilize the entire system.

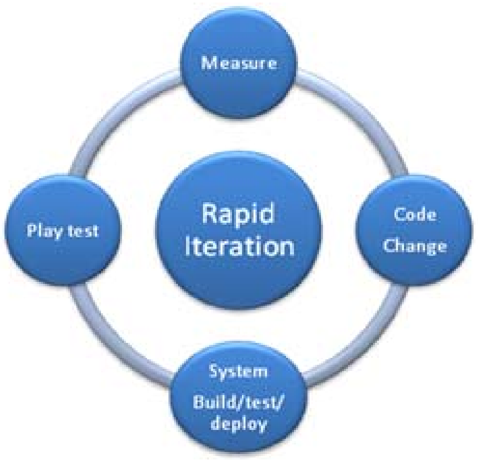


Figure 2: The Rapid Iteration Cycle

The faster and more reliable your MMO can pass through a Full Rapid Iteration Cycle, the more chances you will have of finding the elusive fun factor that will set you apart in the market place. Rapid iteration also helps live operations find and fix critical failure points.

Example: Operations Metrics

Operational metrics span across players, performance and pipeline categories. While the numbers are generally easy to collect, many operational costs require aggregating several numbers to design an actionable metric. For example, take the number of customer service calls, their length and common tasks, and then automate frequent or time-expensive tasks. Aggregating the above numbers and others into a different view allows you to hunt for different classes of players, such as griefers [1] and gold

[1] Griefer = a player who enjoys harassing other players, hurting customer retention and increasing customer service calls.

farmers[2], or any class of player you wish to attract or repel from your game: remember, a customer who generates one half-hour CS call per month has probably wiped out all your month's profit from that customer! Conversely, some types of customers generate far more profit per month than hassle. If you can identify those customers, marketing will find a way to attract them! To get the above aggregate metrics you'll need to track: number of times a player is involved in any customer service complaint, average gold production rate (by character, by time, via IP address and by customer credit card numbers). *Collecting and analyzing such data across multiple sources takes time and money, but generally returns many times the investment rate by improving your player pool*. Finally, analyzing your player's game-specific actions inside your game gives you invaluable pointers into where to evolve your content: from simple game tuning changes to make underutilized content more popular or making friends easier, to examining the net profit from players using micro transactions versus players using a subscription model.

Player Behavior

Measuring what players do in the context of your game is critical to improve your game play and the social binding in the player community. Data points such as popular levels, what objects are most frequently purchased, time to level up and class/weapon balancing are direct pointers to help improve gameplay; whereas data mining forums, friend connectivity and your CS database tell you more about your players as a population or allow you to track down networks of gold farmers. In many ways, measuring player behavior will have greatest impact in making your game fun, but not necessarily profitable if the operational costs are too high.

While obviously there will be many metrics specific to a particular game, there are many others like the above metrics that are common across game types. Player behavior cannot be properly covered in one paragraph: please see http://maggotranch.com/biblio.html for additional papers on metrics.

Where Do We Go from Here?

The sample metrics enumerated above are common to most MMO's but of course there will be a large number of metrics that are highly specific to individual games. To extend, correct and standardize these 'common metrics', an IGDA[3] Special-Interest Group for MMO metrics would aggregate industry experts in the standardization process. But perhaps more importantly, we need to eliminate the dearth of industry expertise in applying metrics from the MMO developers who have successfully done so: what metrics they have found useful, how they captured them and what they were used for would be invaluable data to advance our industry as a whole. I would be happy to invest more of my time in such a SIG: I hate trying to build a large, complex task with the moral equivalent of a flint knife and goat entrails to guide my way. Who's with me?

[2] Gold farmers = groups of people mindlessly playing the game (often with robotic assistance) to acquire gold and other valuable in game items, usually for resale. This tends to throw off economy balance.

[3] IGDA = International Game Developers Association

Conclusion

Unpredictable performance, a lack of player behavior analysis, and the slow iteration rate of MMO's frustrates game designers and managers, and ultimately increases the project risk. With automation, structural code and process changes, we can return to the highly iterative build and test cycles possible in a small code mass environment, with the data from play testers sitting right in front of you. We can even keep that iteration rate up as the system scales, using parallel automation in the build, test, deploy tasks to maintain or accelerate the project's iteration rate.

Yet the biggest roadblock to our industry reaping the rewards of metrics is in fact our own business practices: if a task does not directly "put pixels on the screen", it is not a game feature and thus it is at the bottom of the funding list. Over the past 20 years of building parallel simulations, military training virtual worlds and MMO games, I can't think of a single project that started with metrics tools, despite their proven success in understanding and surviving large-scale software projects. Instead, we always wait until the project is demonstrably stalled. Then and only then is a metrics system, or any other expensive tool, acquired to help get the project moving forward again.

Online games have unprecedented access to more data with higher accuracy than any other service business. Harrahs, a worldwide casino company, used metrics to help them tune their money funnel, winning a national award for optimizing operations and were quoted as saying "metrics are the best business decision we have ever made." If they can do that with less data than we can extract from an online game, an MMO should be a license to print money!

About the Author

Larry Mellon is a consultant, author and speaker on accelerating game production via automated testing and metrics-driven development. Larry joined the game industry over a decade ago, bringing a diverse background in parallel simulation and distributed systems to bear on the development challenges of EA's The Sims Online and The Sims 2.0. Larry then teamed up to help form Emergent Game Technologies: a flexible game development platform grown to worldwide, cross-genre deployment. Before joining the game industry, Larry was a lead architect in DARPA's Advanced Distributed Simulation and Synthetic Theatre of War programs, and a key contributor to the High Level Architecture and RTI 2.0, the standards for integrating all military simulations.

A graduate of the University of Calgary and Project Jade's distributed and parallel computing work in the eighties and nineties, Larry now lives in the San Francisco Bay area.