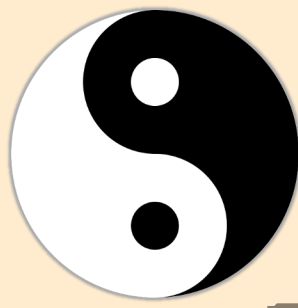


January 2010



IndieZen

*Milestone: Version 0.6.3*

*Create code generators*

*Input = Game Builder data*

*Output = Lua and C++ classes for game entities.*

*Milestone: Version 0.6.4*

*Add networking and integrate Zen Worlds with Game Builder and World Builder.*

*Milestone: Version 1.0.0*

*Polish documentation, tutorials, and interfaces. clean up the installation process.*

## Team Building

*Written by Tony Richards*

*Whether you're a beginner, hobbyist, aspiring game developer, or a seasoned veteran, this article provides some great guidelines and hints for creating a great Indie game development team.*

*The article assumes you're putting together a zero or low-budget team to build an Indie game. It answers questions like "Who is the leader?" and gives you great advice on how to keep your team motivated and making constant, steady progress on your game.*

*If you're considering starting a game development project or if you're already part of a team, this article is definitely a must-read.*

*If you wish, we can discuss this in the forums under this topic.*

*Who is the Leader?*

*You of course! You're the one that has all of the ideas, you should lead the project. You should also reap most of the rewards since everyone else is just doing what you ask them to do.*

*Right?*

*WRONG!*

*In order to put together a successful team, you must find a strong leader. This leader must be able to finish the entire project by going solo, (continued on page 3)*

## Zen :: Introduction

*Zen Open Source Software started back in March of 2007. I had just finished wrestling for several months with Fractured Universe as part of a "Create an MMO Game in 90 Days" contest hosted by Dream Games, Inc.*

*The game came in first place, but it obviously wasn't finished. Who could ever finish any MMO in 90 days?*

*We had created a decent amount of content, but we only populated three fairly small zones. When I said "wrestling," I meant it. During whole contest I was wrestling with code, forcing it to do what I wanted. I ripped code out, threw in more code, completely replaced all of the scripts that came with Torque (then engine I had chosen for the game) and wrote all of my own middleware for the "MMO" portions.*

*I used the DGI MMO Kit as a reference, but for the most part I didn't use much of their code. It was nice to have some references for the "bags" and "inventory" GUI's, and the character creation GUI, etc, but mostly I just looked to see how they did it, then did it my own way. Sometimes I copied stuff a bit, but mostly I just wrote my own code.*

*After the contest I went to IMGDC. One of the round tables at that conference was titled "Slaughtering Sacred Cows" which was quite an eye opener. (continued on page 2)*

## Dysfunctions?

*The Five Dysfunctions of a Team*

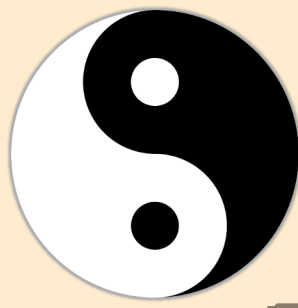
*There are five things to look for when building an Indie team. One of the most difficult points to handle in the Indie Community is the geographic isolation of your members. This makes the next five topics all the more significant when choosing your team members.*

*There are five dysfunctions that we should watch for when building a team. From professional sports to corporate boardrooms to online communities these five personality traits can breakdown cohesive focus and slow productivity to a crawl, and in the Indie community that will kill your project.*

*The first dysfunction is common to all people. Teamwork depends on trust. We must trust our team mates to do what they commit to, and they must trust that we are committed to their satisfaction. When there is no real money involved the currency becomes emotional satisfaction. Therefore, we must be willing to be vulnerable in front of the group. Unless people can be open with their ideas, successes, and failures; it will be impossible to build a foundation of trust.*

*Failure to build trust will lead to the second dysfunction, fear of conflict. (continued on page 4)*

January 2010



IndieZen

## Zen::Introduction

After the contest I went to IMGDC. One of the round tables at that conference was titled "Slaughtering Sacred Cows" which was quite an eye opener. I had never heard of such a term. Now, I knew what it was all about immediately, and it was an interesting approach. Basically, the gist of it was that we can copy other games while making our own, but each element that we put in our game should be put there because we want it there, not simply because the element exists in every other game. That evening, being less of a game designer and more of a software developer, I was mulling over the concept, wondering what other Sacred Cows needed to be slaughtered.

My answer... Indie Game Development Middleware and Tools... I had always taken for granted that Torque and a set of inexpensive tools such as Blender, Quark, Torque Constructor, etc. were my only options when it came to game development. I owned a license for Torque, had the source code for the first three, and knew enough about the products that I could do pretty much anything I wanted to do with them. But, I had always wanted something better. For years (7 years!) I had assumed that the authors of these tools would eventually improve them and make them more powerful, flexible and easier to use. Well, these tools did get better over time, but the improvements were minuscule compared to what I wanted. For starters, Garage Games, the authors of Torque, always heard the complaints about how it was very difficult to make a game using their engine.

"Making games is difficult," was always the answer... or rather it was their excuse for not making Torque easier to use. Sure, when you purchase Torque you get the source code, and anyone that can write a game from scratch can take that source code and make a game. If you're making a game that fits the mold in which Torque was created then quite likely you'll save some time, but if you're making a game that's well outside of that mold then you will end up fighting with Torque more than you end up being creative with it.

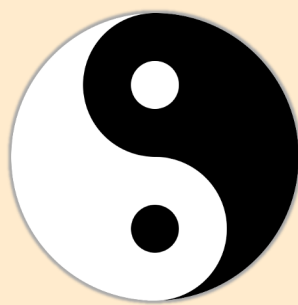
Blender has the same types of problems. The software itself is fairly extensible (if you are fluent

in C, which I'm not), but the models you create with it aren't extensible. There's a new term... "extensible models". My ideal content pipeline is a pipeline in which I can make a bunch of "components" and put them together. Components don't necessarily mean pieces of a building (although that's a good start), and they definitely don't mean "arms" and "legs" of a character model. What I mean is a model that you can take and add animations, morph targets, multiple UV maps, etc, then take the model, choose a subset of the components and export something. An example is something like MakeHuman? or Daz where you can take a humanoid model, apply some modifiers, add some more components like hair, clothing, etc, then export it into a game.

The final assumption, which was yet another cow that needed to be slaughtered, was my assumption that I didn't have time to write these tools and it would take me significantly less time to use the existing tools rather than write my own. By default, that sounds like a fairly sound conclusion, it included two other assumptions... I had been assuming that I would be the only person writing the tools and I had been assuming that I would be the only person using the tools. What if this wasn't the case? What if I could come up with some way to allow hundreds of people to collaborate on making a game engine and some game development tools? What if those hundreds of people could then turn around and make their own games with these tools and each of them save a significant amount of time?

By making a framework of frameworks made specifically for use in game engines and game development tools, I could solve the problem of extensibility. By making it open source and free, anyone can use it. The more people that use it, the more people will extend it, add more features and improve it. The better it gets and the more features it has, the more people will want to use it. And thus the snowball begins rolling down the hill, gathering momentum and more snow.

And that is how IndieZen was born, and the Indie 2.0 Revolution began.



## Indie Developer's Guide - Team Building

### Choose a leader

In order to put together a successful team, you must find a strong leader. This leader must be able to finish the entire project by going solo, although he might not be able to do certain aspects as well as someone else. Generally this is a person that's a skilled programmer, artist and/or game designer that's also a dabbler in the other three subjects. Reserve the "project leader" title for when you have a fairly large team with many people wearing the same hats and it gets to the point where it's obvious that you need a designated leader... and by that time it should be obvious who that leader is. The person doing the bulk of the work and pushing things forward is the leader. The leader should lead by example. He pushes forward and gets things done. He blazes the trail and sets the direction and everyone else follows. Other "management" styles won't work...someone dictating what needs to be done is a dictator, not a leader. A good idea is to start things out as a democracy and once the team gets large enough you elect a leader. This especially works when everyone on the team has read this article and agrees with the bulk of it.

### Avoid Position Titles

Position titles like Programmer, Scripter, Artist define one hat that a person wears. If you put a hat on a person, give him a title, he's less likely to wear multiple hats. Encourage everyone on your team to wear as many hats as they possibly can wear, with one exception... the game designer. Although the game designer should be encouraged to wear other hats too, there should only be one person as the designated game designer. I'll talk about this in more detail in the next section. Also, avoid "Lead Developer", "Lead Artist" for as long as possible. If you have only one person doing all of the programming and he's a good programmer, feel free to let him call himself "senior programmer" but avoid and discourage "lead programmer" type titles... as soon as you recruit another programmer, what happens if he's a better programmer and able to put more time into the project? Demoting someone from "lead" positions is difficult at best... and many times results in morale issues and team breakups. If you need a designated leader for a given subject area, it's best to resort to democracy, with those in that subject area having votes that weigh more than the rest of the team's votes. Complicated, I know, but it's better to let the programmers elect their leader without the artists weighing in too heavily on the vote, otherwise it becomes a popularity contest and ends up electing the most likable person, not the most qualified person.

### One and Only One Game Designer

A game design created by a committee (or worse, created by the community) is a disaster waiting to happen. Too many times I've seen Indie / hobbyist games with a game design put together by a committee of inexperienced game developers, and the game flounders and drags on forever and nothing ever really gets done. You should have one and only one game designer. That game designer should have absolutely zero input from anyone on the game design. He can collaborate with people writing back stories, art, etc, but he shouldn't seek input from them on game mechanics, etc. Ok, now you're thinking I'm crazy and completely wrong... but go with me on this for a minute. Game designs are crapshoots. Even great game designers create game designs that flop, and the only real way you can tell if a game design is fun is to play it.

Keep the game design minimal and implement it as soon as possible.... then and only then should the game designer ask for feedback, thoughts and comments. Feedback should be based on actual game-play experience.

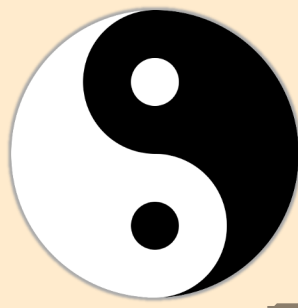
### Ship Early, Ship Often

As soon as you have a playable albeit incomplete demo, ship it internally to all of the developers and let them play it. Even if it's not finished. Even if it has nothing more than a splash screen and a simple scene with a camera flying around. Even if it's an MMORPG and you don't even have multiplayer aspects finished. And once you ship something internally, continue distributing patches and updates... try to automate the process so the updates can happen weekly or at a minimum once a month. When people see something new, they are motivated to contribute more. Team members will motivate other team members this way.... try to get your team to realize that it's more important to develop iteratively and do an internal publish of an unfinished product than it is to never have something visible to show. Enthusiasm feeds enthusiasm, and shipping early also lets other members of the team play the game and provide feedback to the game designer earlier rather than later. When you play a game, even if it's not finished and even if the artwork looks like crap, you can tell if it's going to be fun or not... it's better to find that out as soon as possible. It also gives the game designer and other team members the ability to identify emergent gameplay... typically mistakes or unexpected and unplanned features of the game can be a whole lot more fun than the originally planned game design. Take advantage of it, it's well worth the pains of having to create your distribution and patch system up front. Your goal should be to have an internally distributable game within 30 days of starting the project, and you should have major features implemented within the first 90 days. Fix Bugs Immediately While features can remain broken and incomplete, don't stand for bugs that cause application crashes or other instabilities. Fix them immediately, otherwise the benefits of "Ship Early and Ship Often" will be negated. Have a bug tracking system that everyone on the team can use, and every time there's any kind of bug have them enter it.

### Version Control and Branching

There's nothing more important to a game development project than version control and branching. All new features should be implemented in a branch, while the trunk of your version control system remains your "golden stable". You should take advantage of peer code reviews and peer art reviews... the person or people who do the work should not be the person who merges the branch into trunk and do the reviews. Keep branches short lived. While you can distribute an internal distribution from a branch to test a new feature, it's not a good idea to have multiple branches being developed simultaneously for an extended period of time. Don't forget to create tags for every release, whether it's internal or not. You never now when you're going to need to be able to reproduce a given branch.

by Tony Richards  
CTO IndieZen.org



## The Five Dysfunctions of a Team

To make a product, game or otherwise, all the details must be worked out. Everything from the color pallet to the target market, must be discussed. Passionate debate is necessary to ensure that, at the end of the day, the best product is on the drawing board. Teams that lack trust will avoid debate and turn instead to veiled discussions and guarded comments, thus undermining the team.

Without healthy conflict we can ensure the third dysfunction will visit the team, lack of commitment. Active debate allows everyone to be heard and inspires a feeling of ownership in the product and a feeling of responsibility to the team. These two powerful forces make up the basis for buy in and commitment. Without it the team may feign agreement during meetings but are unlikely to buy in and commit to decisions.

Without any real commitment an buy in, team members develop an avoidance of accountability, the fourth dysfunction. With no commitment to a clear plan, even the most dedicated and focused team members will hesitate to call their peers out on issues and behaviours that seem counter-productive to the good of the team.

When the team fails to hold its members accountable the fifth dysfunction will begin to fester, inattention to results. Without results there is no product and no satisfaction for the team. This inattention to results comes from members putting their individual needs (such as ego, recognition, or status) or the needs of their departments above the goals of the team.

To put a positive spin on it, look at developing the opposites of these dysfunctions. With the goal of a cohesive team we can define the correct behaviours:

- 1: Develop trust among the team members.
- 2: Encourage unfiltered, passionate debate.
- 3: Seek firm commitment to the decisions and plans of action.
- 4: Hold members accountable to one another for for delivering on commitments.
- 5: Focus on the overall achievement of collective results.

It is a simple plan. One that takes a level of discipline, persistence, and patience that few teams can muster.

—find it on [amazon.com](http://amazon.com)—

The Five Dysfunctions of a Team  
by Patrick Lencioni