# The Trouble with Threat Modeling

Adam Shostack here.

I said recently that I wanted to talk more about what I do. The core of what I do is help Microsoft’s product teams analyze the security of their designs by threat modeling.   So I’m very concerned about how well we threat model, and how to help folks I work with do it better.   I’d like to start that by talking about some of the things that make the design analysis process difficult, then what we’ve done to address those things.  As each team starts a new product cycle, they have to decide how much time to spend on the tasks that are involved in security.  There’s competition for the time and attention of various people within a product team.  Human nature is that if a  process is easy or rewarding, people will spend time on it.  If it’s not, they’ll do as little of it as they can get away with.  So the process evolves, because, unlike [Dr No](http://blogs.msdn.com/sdl/archive/2007/08/30/dr-no-and-risk-management.aspx), we want to be aligned with what our product groups and customers want

There have been a lot of variants of things called “threat modeling processes” at Microsoft, and a lot more in the wide world.   People sometimes want to argue because they think Microsoft uses the term “threat modeling” differently than the rest of the world.  This is only a little accurate.  There is a community which uses questions like “what’s your threat model” to mean “which attackers are you trying to stop?”  Microsoft uses threat model to mean “which attacks are you trying to stop?”  There are other communities whose use is more like ours.  In this paragraph, I’m attempting to mitigate a denial of service threat, where [prescriptivists](http://www.thefreedictionary.com/prescriptivist) try to drag us into a long discussion of how we’re using words.)   The processes I’m critiquing here are the versions of threat modeling that are presented in[*Writing Secure Code*](http://www.amazon.com/Writing-Secure-Second-Michael-Howard/dp/0735617228/ref%3Dpd_bbs_sr_1/002-3554229-7012008?ie=UTF8&s=books&qid=1190834155&sr=8-1)*,* [*Threat Modeling*](http://www.amazon.com/Threat-Modeling-Microsoft-Professional-Swiderski/dp/0735619913/ref%3Dpd_bbs_8/002-3554229-7012008?ie=UTF8&s=books&qid=1190834155&sr=8-8), and [*The Security Development Lifecycle*](http://www.amazon.com/Security-Development-Lifecycle-Michael-Howard/dp/0735622140/ref%3Dpd_bbs_5/002-3554229-7012008?ie=UTF8&s=books&qid=1190834155&sr=8-5) books.

In this first post of a series on threat modeling, I’m going to talk a lot about problems we had in the past.  In the next posts, I’ll talk about what the process looks like today, and why we’ve made the changes we’ve made.   I want to be really clear that I’m not critiquing the people who have been threat modeling, or their work.  A lot of people have put a tremendous amount of work in, and gotten some good results.  There are all sorts of issues that our customers will never experience because of that work.  I am critiquing the processes,  saying we can do better, in places we are doing better, and I intend to ensure we continue to do better.

We ask feature teams to participate in threat modeling, rather than having a central team of security experts develop threat models.  There’s a large trade-off associated with this choice.  The benefit is that everyone thinks about security early.  The cost is that we have to be very prescriptive in how we advise people to approach the problem.  Some people are great at “think like an attacker,” but others have trouble.   Even for the people who are good at it, putting a  process in place is great for coverage, assurance and reproducibility.  But the experts don’t expose the cracks in a process in the same way as asking everyone to participate.

### Getting Started

The first problem with ‘the threat modeling process’ is that there are a lot of processes.   People, eager to threat model, had a number of TM processes to choose from, which led to confusion.  If you’re a security expert, you might be able to select the right process.  If you’re not, judging and analyzing the processes might be a lot like analyzing cancer treatments.  Drugs?  Radiation?  Surgery?  It’s scary, complex, and the wrong choice might lead to a lot of unnecessary pain.   You want expert advice, and you want the experts to agree.

Most of the threat modeling processes previously taught at Microsoft were long and complex, having as many as 11 steps.  That’s a lot of steps to remember.  There are steps which are much easier if you’re an expert who understands the process.  For example, ‘asset enumeration.’  Let’s say you’re threat modeling the GDI graphics library.  What are the assets that GDI owns?  A security expert might be able to answer the question, but anyone else will come to a screeching halt, and be unable to judge if they can skip this step and come back to it.  (I’ll come back to the effects of this in a later post.)

I wasn’t around when the processes were created, and I don’t think there’s a lot of value in digging deeply into precisely how it got where it is.  I believe the core issue is that people tried to bring proven techniques to a large audience, and didn’t catch some of the problems as the audience changed from experts to novices.

The final problem people ran into as they tried to get started was an overload of jargon, and terms imported from security.  We toss around terms like repudiation as if everyone should know what it means, and sometimes implied they’re stupid if they don’t.  (Repudiation is claiming that you didn’t do something.  For example, “I didn’t write that email!,” “I don’t know what got into me last night!”  You can repudiate something you really did, and you can repudiate something you didn’t do.)  Using jargon sent several unfortunate messages:

1. This is a process for experts only
2. You’re not an expert
3. You can tune out now
4. We don't really expect you to do this well

Of course, that wasn’t the intent, but it often was the effect.

### The Disconnected Process

Another set of problems is that threat modeling can feel disconnected from the development process.  The extreme programming folks are fond of only doing what they need to do to ship, and Microsoft shipped code without threat models for a long time.  The further something is from the process of building code, the less likely it is to be complete and up to date.  That problem was made worse because there weren’t a lot of people who would say “let me see the threat model for that.”   So there wasn’t a lot of pressure to keep threat models up to date, even if teams had done a good job up front with them.  There may be more pressure with other specs which are used by a broader set of people during development.

### Validation

Once a team had started threat modeling, they had trouble knowing if they were doing a good job.  Had they done enough?  Was their threat model a good representation of the work they had done, or were planning to do?  When we asked people to draw diagrams, we didn’t tell them when they could stop, or what details didn’t matter.  When we asked them to brainstorm about threats, we didn’t guide them as to how many they should find.  When they found threats, what were they supposed to do about them?  This was easier when there was an expert in the room to provide advice on how to mitigate the threat effectively.   How should they track them?   Threats aren’t quite bugs—you can never remove a threat, only mitigate it.  So perhaps it didn’t make sense to track them like that, but that left threats in a limbo.

### "Return on Investment"

 The time invested often didn’t seem like it was paying off.  Sometimes it really didn’t pay off.    (David LeBlanc makes this point forcefully in “[Threat Modeling the Bold Button is Boring](http://blogs.msdn.com/david_leblanc/archive/2007/09/19/threat-modeling-the-bold-button-is-boring.aspx)”) Sometimes it just felt that way—Larry Osterman made that point, unintentionally in “[Threat Modeling Again, Presenting the PlaySound Threat Model](http://blogs.msdn.com/larryosterman/archive/2007/09/17/threat-modeling-again-presenting-the-playsound-threat-model.aspx),” where he said “Let's look at a slightly more interesting case where threat modeling exposes an issue.”  Youch!  But as I wrote in a comment on that post, “What you've been doing here is walking through a lot of possibilities.  Some of those turn out to be uninteresting, and we learn something.  Others (as we've discussed in email) were pretty clearly uninteresting”  It can be important to walk through those possibilities so we know they’re uninteresting.  Of course, we’d like to reduce the time it takes to look at each uninteresting issue.

### Other Problems

Larry Osterman lays out some other reasons threat modeling is hard in a blog post: <http://blogs.msdn.com/larryosterman/archive/2007/08/30/threat-modeling-once-again.aspx>

One thing that was realized very early on is that our early efforts at threat modeling were quite ad-hoc.  We sat in a room and said "Hmm, what might the bad guys do to attack our product?" It turns out that this isn't actually a BAD way of going about threat modeling, and if that's all you do, you're way better off than you were if you'd done nothing.

Why doesn't it work?  There are a couple of reasons:

It takes a special mindset to think like a bad guy.  Not everyone can switch into that mindset.  For instance, I can't think of the number of times I had to tell developers on my team "It doesn't matter that you've checked the value on the client, you still need to check it on the server because the client that's talking to your server might not be your code.".

Developers tend to think in terms of what a customer needs.  But many times, the things that make things really cool for a customer provide a superhighway for the bad guy to attack your code.

It's ad-hoc.  Microsoft asks every single developer and program manager to threat model (because they're the ones who know what the code is doing).  Unfortunately that means that they're not experts on threat modeling. Providing structure helps avoid mistakes.

With all these problems, we still threat model, because it pays dividends.  In the next posts, I’ll talk about what we’ve done to improve things, what the process looks like now, and perhaps a bit about what it might look like either in the future, or adopted by other organizations.

# The New Threat Modeling Process

Adam Shostack here, with the second post in my series on the evolved threat modeling process. To summarize, what I’ve tried to achieve in changing the process is to simplify, prescribe, and offer self-checks. I’ll talk in the next post about why those three elements are so important to me. For now, let me describe the process.

One of the largest changes that we’ve made is to a simplified process (and diagram). I like to say that this looks pretty much like every other software process diagram you see today. That’s intentional. There’s only so much we can expect people to take away from a class, and making this simple and familiar helps ensure there’s room for the other important parts.

First, the “[process hamster wheel](http://www.securitymetrics.org/content/Wiki.jsp?page=Welcome_blogentry_061005_1),” (with apologies to Yankee Group analyst Andy Jaquith): **

Change

Now that you’ve seen the wheel, I’ll briefly describe the steps:

1. **Vision**: Consider your security requirements, scenarios and use cases to help frame your threat modeling. What are the security aspects of your scenarios? What do your personas expect or hope doesn’t happen? What are the security goals of the system you’re building, and how do those interact with the system as it stands?
2. **Model**:

The basic idea is to create a diagram of your software, showing all trust boundaries.

* 1. Draw a diagram of your software. We encourage use of the DFD formalisms, which Larry Osterman describes in this post. <http://blogs.msdn.com/larryosterman/archive/2007/08/30/threat-modeling-once-again.aspx> Essentially, the elements are
		1. External entities (anything outside your control)
		2. Processes (running code)
		3. Data stores (files, registry entries, shared memory, databases)
		4. Data flows (which connect all the other elements)
	2. Draw trust boundaries between components. You can do this on a whiteboard, in Visio, or in one of the specialized threat modeling tools we’ve built. (A trust boundary is anywhere that more than one principal can access an object, such as a file or process.)
	3. If your trust boundary crosses something which isn’t a data flow, you need to break it into two logical elements, or draw a sub-diagram with more details. (This is different advice: we used to tell people trust boundaries could only cross data flows. People drew them anywhere that felt right. We decided to go with what people were doing—there was important information in what they were expressing.)
	4. If you need more details to express where trust boundaries are, add an additional diagram.
	5. When you don’t have any more trust boundaries to draw, you’re done.
	6. If a diagram doesn’t have any trust boundaries, you’ve gone too far.
1. **Identify Threats using STRIDE/element**

For each element in your diagram, consider threats of the types indicated in this chart. (We’ll come back to the chart’s origins in a later post.)



There’s an important mis-conception we often see, which is that STRIDE is appropriate for use as a classification system. It’s really hard to use STRIDE to describe attacks—the impacts blend together really quickly. The most valuable use of STRIDE is to help people think about how threats have impacted elements of a design in the past. That is, it’s a valuable framework for finding threats, not for describing them. “What if someone spoofs this host?”

1. **Mitigate**

Here on the SDL strategy team, we love threat modeling. We know that not everyone feels that way, and we ask teams to threat model so that they can find and ***mitigate*** threats. A threat model document with great diagrams and lots of threats isn’t very useful if you don’t take the key step of addressing the issues you find. There are four ways to do that:

* 1. Redesign to eliminate threats.
	2. Use standard mitigations, such as those provided by OS features, to mitigate threats. Examples of this include ACLs to protect files from tampering, cryptography to protect data flows between machines, or DPAPI to protect secrets.
	3. Invent new mitigations, understanding that this is a subtle art.
	4. Accept risk, when allowed by the SDL
1. **Validate**

There are two levels of validation. The first is within each stage, the second is a validation pass at the end of the process. That end of process validation entails:

* 1. Make sure that the diagrams are up-to-date and accurate
	2. Ensure that you have STRIDE threats per data flow that crosses a trust boundary, and for each element that such a trust boundary connects to
	3. Make sure you’re mitigating each threat
		1. You have a bug filed per threat that you want to mitigate. The bug should be of the form “attacker can do X. Proposed fix: Y.” You might include tradeoffs you’re making, and possibly have test plans in the bug, if you include those.
		2. You have a valid reason for each non-mitigated threat not being mitigated/
		3. All threats are in class i or ii.

**5.a. On change, re-validate**

This hamster wheel has a very intentional brake on it: the word change, above validate. What that means is you want to go through the process again when you make changes that need to be on the diagram. Checking to see if your diagrams change is a relatively simple check that allows people to track changes against their threat model as their design iterates.

In the next article, I’ll talk about the reasoning behind the design, and offer up some process tools that anyone can use to make a process more user-friendly.

# [Getting into the Flow With Threat Modeling](http://blogs.msdn.com/sdl/archive/2007/10/11/getting-into-the-flow-with-threat-modeling.aspx)

Adam Shostack again, with the third in our series on threat modeling. In this post, I want to explain one of the ‘lenses’ that seemed to help us focus threat modeling, and how I’ve applied it.

The concept of [flow](http://en.wikipedia.org/wiki/Flow_%28psychology%29) originated with Mihaly Csikszentmihalyi. It refers to a state where people are energetically involved with what they’re doing. Seeing this a few times during threat modeling sessions made it obvious when it was missing, and it was missing often. I set out to address some of the elements that seemed to make threat modeling harder. The Wikipedia article (currently) has a good list, so I’ll focus in on a few of them:

1. Clear goals
2. Direct and immediate feedback
3. Balance between ability and challenge
4. Focused attention

Let’s take these one at a time.

### Clear Goals

Giving people clear goals is important because it helps take them from worrying about what your goals mean to worrying about how to achieve them. Without clear goals, it’s very challenging to get into the spirit of anything, whether playing a game or shipping an operating system. As goals go, “think like an attacker” and “brainstorm” aren’t up there with “A PC on every desktop.” The lack of a clear answer to “how do I know I’m done with my diagrams” or “how many threats should I have,” or “what is a good threat model” made it hard for people to do well, and just as importantly, even people who were doing well weren’t sure that they were doing well. Which brings us directly to the next point:

### Direct and Immediate Feedback

When people have direct and immediate feedback, they can adjust their activities relatively easily.  If you’re cooking from a well written recipe, at each stage, you have some indicators that things are done.  “Are the onions brown?”  “Is the water boiling?”  “Does the pasta stick to the wall?”  This is true of any skill we’re learning.  With threat modeling, when there is no expert in the room, feedback cantake weeks or months to come back. As we rolled threat modeling out at Microsoft, it was possible for an entire threat model to be cooked without any course correction. We’re doing better now - there are more experienced people around, better documentation, and more focus on how to self-check. But at one time we asked people to engage in really complex tasks from the get go, without trying to achieve a…

### Balance Between Ability and Challenge

Almost by definition, when you’re new at something, you don’t have a lot of skill around it. If the new activity is something highly complex, like speaking a foreign language, you’re likely to be anxious. And once you are skilled, simple tasks, like talking to a beginner, are often boring. Which brings us to the chart, which I’ve redrawn from Csikszentmihalyi’s book, Flow.    [](http://blogs.msdn.com/blogfiles/sdl/WindowsLiveWriter/GettingintotheFlowWithThreatModeling_96D3/image_2.png)

The ideal learning experiences involve a balance of challenge and reward, each of which grows as you learn. For example, when you buy Halo, there’s a training mission that allows you to run around doing simple tasks, learning the controls. If we dropped people into the live online game without any practice, they’d get slaughtered, and that’s bad for everyone. The [newb](http://en.wikipedia.org/wiki/Newbie) is frustrated, and the experienced player is bored. (In fact, there’s pretty explicit acknowledgement of this in the [Wired article on Halo](http://www.wired.com/gaming/virtualworlds/magazine/15-09/ff_halo).)

So looking at the chart of skills versus challenge, there’s a channel in the middle—it’s the flow channel. We’d like to guide people there, by giving them challenges that are in line with their skills. Once people are there, we want to prod them from point A to point D, where there are larger challenges, and skills that allow people to address those challenges. Sometimes, we want them to leave their comfort zone and learn new things. To take on new challenges (going to B) and learn from them, going to D. If we don’t challenge people, they get bored, and it’s less likely that they’ll learn. (This obviously isn’t a complete model of learning, but it flows out of the discussion.)

We want to do that for Halo, and we want to do that for threat modeling. To date, this has been implicit. We’ve simplified the process, in part in the interest of getting people out of the anxiety zone, and into a place where they can feel comfortable and then learn. Over time, we’ll be challenging and encouraging people to do more complex tasks as part of SDL threat modeling.

This assumes that threat modeling is a frequent enough activity that people develop skills around it. I think that’s an ok assumption, as long as we design good refresher and update training and documentation.

### Focused Attention

If you’re reading this blog post while talking on the phone and petting your dog, you’re going to get less out of it than if you do nothing else while reading it. Similarly, when you’re learning a new skill or performing a hard task, you need to focus your attention. Related to this is moving around. Getting a little blood flow helps you be energetic and engaged. This has been one of the hardest elements of flow to add to threat modeling. Many people here multi-task intensively, with laptops or blackberries always at hand. Challenging that habit seems like a challenge to the way people work, and I’ve shied away from issuing that challenge.

### Why flow?

One final point I’d like to address is that flow is one of many frameworks we could use to address problems in threat modeling. My choice to use it was a driven by its striking absence. Flow isn’t the only framework for thinking about these things, and there are some criticisms associated with trying to shoehorn everything into flow. For threat modeling, it just seemed to well…flow.

# [Making Threat Modeling Work Better](http://blogs.msdn.com/sdl/archive/2007/10/16/making-threat-modeling-work-better.aspx)

Adam Shostack here, with part four of my threat modeling series. This post is a little less philosophical and a lot more prescriptive than the one about flow. It explains exactly how and why I changed a couple of elements of the process. The first is the brainstorming meeting, and the second is the way trust boundaries may be placed.

The brainstorming meeting is a mainstay of expert threat modeling. It’s pretty simple: you put your security experts in a room with system diagrams and a whiteboard. Usually, you put your system designers in there, and make them promise not to strangle your experts. Optionally, you can add beer or scotch. Sometime later, you get a list of threats. How long depends on how big the system is, how well its requirements are documented, and how well your experts work together.

We like having our developers threat model. There are a lot of reasons for this. Not only do they know the system better than anyone else, but getting people involved in a process helps ensure that they buy into it.

Now this desire is great, but it leads to some issues, first and foremost is that many of the people who are now involved aren’t security experts. This means that they lack both direct experience of the process and the background that informs it. This isn’t a slam on them. I lack experience in the database design process, and I don’t have years of experience to help orient me. So I’d make mistakes designing a database, and someone who isn’t a security expert may make mistakes in security. For example, someone might try to use encryption to mitigate tampering threats. (The SDL crypto requirements cover this, and I try to gently correct them to integrity mechanisms like MACs or signatures.) This is a reality that we have to account for at the process design level.

### Adding Structure to Chaos

So how does this relate to the brainstorming meeting? It’s a dramatic increase in the need for structure. Where experts may think they do better threat modeling with scotch in hand, , it certainly doesn’t lead to beginners having a flow experience. So we need a structure, and we need to provide it.

We encourage people to get started by drawing a whiteboard diagram. Almost everyone in software draws on whiteboards regularly, and this makes it **an ideal first step.** It’s an ideal first step because everyone can do it, see that they’ve done it, and feel like they’re making progress.

The core mechanism we’ve used to provide it is the STRIDE/element chart. (I’ll talk a lot more about its origins and limits in a few posts, but for now, let’s pretend it’s gospel, and enumerates all possible threats.) Given this gospel, it becomes possible to step through the threat modeling diagram, “turn the crank,” and have threats come out. “Item 7 is a data flow? Let’s look for T,I and D.” (Tampering, Information disclosure, and Denial of service.)

Similarly, we have four ways of addressing threats – redesign, standard mitigations, new mitigations, and risk acceptance. We have training on mitigating threats, we have explanation of why and when to use each (and they’re presented in a preferred order).

Lastly, we provide advice about how to validate the threat model and it’s relation to reality.

Between these four steps and the hamster wheel which ties them together, we give people the structure in which they can take on the process. The other thing I wanted to address is how we respond to consistent “errors” that we see.

### Where Trust Boundaries Show Up

We used to give people clear guidance that trust boundaries should only intersect with data flows. After all, you can’t really have a process that’s half-running as admin, and half as a normal user. Logically, you have two entities. And people kept drawing trust boundaries across processes and data stores. It drove me up the wall. It was *wrong.*

As people kept doing it, I decided to swallow my pride and accept it. I now tell people to put their trust boundaries wherever they believe one exists. And they’ve continued exactly as before, but I’m a lot happier, because I’ve found a way to help them draw more detailed diagrams where they need them. Which includes anywhere a trust boundary crosses a process or data store. They’re happier too. No one is telling them that they’re wrong.

I was going to title this post “Lord grant me the strength to change the things I can, the courage to accept what I can’t, and the wisdom to know the difference,” but, first, it’s too long, and second, if we started that way, it would be wrong to add beer or scotch.

# [Threat Modeling Self Checks and Rules of Thumb](http://blogs.msdn.com/sdl/archive/2007/10/22/threat-modeling-self-checks-and-rules-of-thumb.aspx)

Adam again. I hope you’re still enjoying this as we hit #5 in the threat modeling series.

In my last post, I talked about how almost everyone in software draws on whiteboards regularly, and this makes it **an ideal first step.** It’s an ideal first step because everyone can do it, see that they’ve done it, and feel like they’re making progress.

That wasn’t quite complete. Not only do we want people to see that they’ve done it, we want to give them a way to validate their work or other people’s work. So we ask them to tell a story. We’re not asking for Shakespeare here, we’re asking them to explain how their software will be used, and to make sure that their diagram supports that story, and that it relates to their actual software.

We also give them rules of thumb (lots of rules of thumb) about things we often see wrong in diagrams:

* Don't have data sinks: you write the data for a reason. Show who uses it.
* Data can’t move itself from one data store to another: show the process that moves it.

(Larry Osterman has some in his blog post, "[Threat Modeling Rules of Thumb](http://blogs.msdn.com/larryosterman/archive/2007/09/21/threat-modeling-again-threat-modeling-rules-of-thumb.aspx)" I helped edit those, but want to suggest additional changes. In particular, “you need to be concerned” is not actionable. “Review this carefully,” or “Focus your attention here” are more actionable. People threat modeling are already concerned.)

Good “rules of thumb” encourage flow by empowering people to make a snap decision and move along.

# [The STRIDE per Element Chart](http://blogs.msdn.com/sdl/archive/2007/10/29/the-stride-per-element-chart.aspx)

I’d like to talk about the STRIDE per element chart in the sixth post of my threat modeling series. I’d like to talk about where it’s from, some of the issues that come with that heritage, and how you might customize it in your own threat modeling process.

Michael Howard and Shawn Hernan did an analysis of our bulletins and some CERT and CVE data.  Their goal was to validate work they'd done on threat trees.  (Covered in the SDL book.)  They were *looking for classes of things that would cause us to ship updates.* That’s tremendously important, so I’ll repeat it. They were looking for classes of things that would cause us to ship an update.  If we wouldn’t update for it, *it doesn’t exist in the chart.* That’s not to say it doesn’t exist. If there’s an elevation of privilege against an external entity, well, by definition, we can’t fix it. It’s external. So is there value in calling out that risk in threat modeling? Sometimes there might be.

### Breach Disclosure Laws

Sometimes organizations lose control of data that’s been entrusted to them. You might find that risk as an instance of “information disclosure by external entity.” So should we have a check there? Well, on the one hand, we don’t have any control over it - unless we hand them that information by design. In which case, perhaps we could design to hand over less.  At Microsoft, we haven't added it to the chart we ask people to use, and we'll revisit that over time.

### Specific Elements

Another thing you might note is that the STRIDE chart is sorta vague. A process could be an exe, a .NET assembly, or an a.out executable running on Unix v7. Each of those will be vulnerable to different instantiations of threats. Your exe or a.out will be vulnerable to simple stack smashing overflows, but the .NET assembly won’t be. As you make your elements more specific, you can provide more prescriptive guidance as to what threats to look for, and how to effectively mitigate them.

### Customizing the Chart for Your Threats

The chart is centered on our needs at Microsoft. Those may not be your needs. Perhaps you’re building a threat modeling process to focus on voting. You can get much more specific about what a process is, and what threats it might come under. You might have really specific threats against voter lists (a type of data store) or vote tallies (another store).

Being able to enumerate the assets in data stores helps you motivate threats, and assess their risk and importance of fixing them. When we build a threat modeling process for the SDL, it gets used in Windows, Office, SQL, Exchange, Dynamics, and a slew of other products. We need some degree of consistency, so we can deliver consistent products and messages to our customers. We also need to encourage customization and specificity, so that the process is as prescriptive as we can make it. Doing so allows you to make it more prescriptive, appropriate and evocative for your users.

# [Don't Try This At Home](http://blogs.msdn.com/sdl/archive/2007/11/05/don-t-try-this-at-home.aspx)

No, it’s not a post on why Adam should never volunteer to do a 12 part series on threat modeling, but rather, why inventing your own mitigations is hard, and why we suggest treading carefully if you need to go there.

Let me first explain what I mean by mitigations because apparently there’s some confusion. We have folks here at Microsoft who call things like [the /GS compiler flags](http://msdn2.microsoft.com/en-us/library/aa290051%28vs.71%29.aspx) "mitigations." When I talk about mitigations in a threat modeling context, I mean things that prevent an attack from working. For example, encryption mitigates information disclosure threats. If you use a strong cryptosystem, and you keep your keys secret, attackers can’t read your data.

Next, why would you need to go there? Often times you don’t. The problems that we face in building software projects are often similar to the problems that other software projects have faced, and we can learn from their successes and mistakes. There’s been a fair amount of good work done in patterns in software, and we even have design principles that we can use. Sometimes, the work being done is really new and different, and there’s a need to create innovative mitigations.

The security experts in the audience are cringing at the very idea. They know that this often ends in pain. I’d like to explain why.

Designing security mitigations is a skill. If I’m trying to design a database, my personal lack of database skills will be obvious: it will be slow, it might lose data, and we’ll discover there’s a problem very quickly. In contrast, if I’m designing a cryptosystem, I might not be able to break my own design. This is a common situation for amateur cryptographers. They build a system which they can’t break, but when an expert looks at it, it’s no more reliable than [a fish in a blender](http://en.wikipedia.org/wiki/BassOmatic).

Custom mitigations always result in a need for expert analysis, by someone who can draw on years of experience about what goes wrong. Ideally, that expert works for you, and looks at the idea long before it’s implemented… never mind shipping.

One of the critiques of the threat modeling ~~blog posts~~ process is that it can seem interminable. And so, in this final post, I’d like to offer up some final thoughts on language, and cognitive load.

**Specification versus Analysis**

When Larry Osterman was [writing about threat modeling](http://blogs.msdn.com/larryosterman/archive/2007/09/14/threat-modeling-again-pulling-the-threat-model-together.aspx), he casually tossed out:

A threat model is a specification, just like your functional specification (a Program Management spec that defines the functional requirements of your component), your design specification (a development spec that defines the architecture that is required to implement the functional specification), and your test plan (a test spec that defines how you plan on ensuring that the design as implemented meets the requirements of the functional specification).

Just like the functional, design and test specs, a threat model is a living document - as you change the design, you need to go back and update your threat model to see if any new threats have arisen since you started.

I found this pretty surprising. I think of threat modeling as an analysis technique, but hey, if test plans are specs, threat models aren’t that different. (It took some private back and forth with Larry to convince me.) This brings me to the topic of what words we use to describe things.

**On language**

[The English language] becomes ugly and inaccurate because our thoughts are foolish, but the slovenliness of our language makes it easier for us to have foolish thoughts. The point is that the process is reversible. Modern English, especially written English, is full of bad habits which spread by imitation and which can be avoided if one is willing to take the necessary trouble. If one gets rid of these habits one can think more clearly, and to think clearly is a necessary first step toward political regeneration: so that the fight against bad English is not frivolous and is not the exclusive concern of professional writers. (George Orwell, [Politics and the English Language](http://www.mtholyoke.edu/acad/intrel/orwell46.htm).)

We ask people to threat model (as part of the SDL) by threat modeling their application. To do that, they model the design, and then have a threat modeling meeting in which someone runs the threat modeling tool to produce a threat modeling document. In our particular case, I’m not sure it’s easily avoided.

“We ask people to analyze their designs (as part of the SDL). To do that, they diagram the design, then have a threat modeling meeting in which someone runs the *codename* tool to produce a threat modeling document.”

In fact, I’ve worked hard to reduce jargon in the process, but at one point, went too far. I tried to convince people to say “diagram” instead of DFD, and they revolted. Actually, they didn’t really revolt, but they did refuse to go along. Experienced threat modelers actually felt that the term DFD was *more clear* than saying ‘diagram,’ because DFD includes a specification of what kind of diagram, and they had already integrated the acronym into their thinking. Having integrated it, they could use it without thinking about it. Well, [as Adam Barr points out](http://www.proudlyserving.com/archives/2007/11/writing_about_m.html), “you can pick your friends, and you can pick your nose, but you can't pick your online community's lexicon.”

It didn’t add to their cognitive load to say “DFD.”

**Cognitive Load**

People can only handle so much new information at one time. Learning to drive a stick shift is harder than learning on an automatic, because there’s a whole set of tasks you can ignore while the automatic transmission handles them for you. In my approach, this relates pretty closely to the concept of flow. If you’re so focused on rules and jargon, you can’t focus on what you should be building. Cool, well-designed features to help your customers.

**In Conclusion**

We started this series by looking at the trouble with threat modeling. How people had trouble getting started, relating the process to their other work, or validating what they’d done. We’ve looked at the new threat modeling process, and the steps involved. We talked about how to get into the flow with threat modeling. How clear goals, direct and immediate feedback, and balancing ability and challenges set people up to focus their attention and succeed. We’ve talked a bit about making threat modeling work better by adding structure to chaos, and how to use self-checks and rules of thumb to give people confidence they’re on the right trail. We’ve talked a very little bit about how to customize the process for your own needs, and where that customization can be dangerous.

All of this has come out of looking at our threat modeling activity as a human activity, and asking how we can best shape it to help people get to the results that we want. I hope that our readers have enjoyed the series, which we’ve converted into a downloadable document which you’re reading.