A "programming paradigm" is a way of approaching a problem.

With strengths, weaknesses, capabilities, and limits.
Not just a choice of language.
Full disclosure

In theory, I am supposed to be teaching you "how to program correctly", and how to follow the "one true path to programming greatness."

I find this attitude **reprehensible**.

It is based upon a world view that is not simply archaic, but also capable of **damaging your careers**.
Myth #1: the facile programmer can choose the "one true paradigm" for a project in isolation from other factors.
Reality: programming is now a social activity in which the "one true path" is a meaningless concept.
Religious wars

Proponents of one paradigm often engage in *holy wars*: vi versus emacs 
.NET versus LAMP
Google versus Amazon (for clouds).
For languages:
   Functional versus imperative.
   Strong versus weak typing.
   Speed versus robustness.
   Memory models.
   High-level versus low-level.

We must be careful not to descend into this pit.
The proper paradigm is not an absolute; it depends upon
The **social context** of each paradigm
(technological) **Footprint:**
   How many people use it?
   Has it been used successfully in similar cases?
   How many people know how to use it?
(software) **Legacy:**
   What has been used before?
   What lessons were learned?
   What should we do differently?
   Why should we do it differently?
(technological) **Infrastructure:**
   What pre-existing resources are there?
   What new resources are required?
   How much will they cost?
   Can we afford it?
Myth #2: there is a programming paradigm appropriate to solve all programming problems.
Reality: paradigms evolve in order to address specific cases, not the general case.
I have heard quite enough of "survival of the fittest." In the real world, it is "survival of that which fits."

Programmers do not survive by "being the fittest"; they survive by "fitting in" and by getting the job done.

Likewise, paradigms do not survive by "being the fittest". They survive by fitting into the larger context of software development.

The most important job interview advice: "don't be a programming fascist."
The social nature of programming

"Google is your friend."
Examples and success stories.
Advice about common (and not so common) quandaries.
A basic and fundamental paradigm shift

From "survival of the fittest": the programmer is a self-sufficient island. To "survival of those who fit": the programmer is a connected and active citizen of a culture.
How to fit:

Share your expertise and experience. View that sharing not as fact, but as social evidence. Become an intelligent consumer of evidence.
Patterns and rituals

So far, we've learned about patterns: well-documented solutions to common problems.

The internet is populated with **rituals**: poorly documented solutions that worked **once**.

A ritual becomes a pattern after many people have tested it and +1'd it.

Every pattern started out as a ritual.
Alchemy versus chemistry

Alchemy: rituals and half-truths. Things that sometimes work.
Chemistry: patterns and invariants. Things that always work.
Myth #3: one should always choose the most recent paradigm; it will be the easiest to use. Reality: the choice is also based upon other factors, including frequency of adoption and social pressure.

Beattie, Cowan, Wagle, Wright, and Shostack, Timing the Application of Security Patches for Optimal Uptime
<https://www.usenix.org/conference/lisa-02/timing-application-security-patches-optimal-uptime>
When you live on the bleeding edge, you tend to bleed.

Causes of injury include social isolation of first adopters.
   Inadequate accumulated knowledge: you're on your own in facing problems.
   Inadequate social pressure: few adopters means that you will discover most of the defects.
   Inadequate success stories: higher risk of project failure.
   Inadequate infrastructure: missing libraries, compilers, implementations, ...
"The Magic Show"

As long as you follow the steps, you will get the same result. But if you deviate -- in any respect -- you are likely to accomplish nothing at all. These videos are rituals.

This makes good PR, but is incredibly misleading.

So, rituals are useful only as long as all you need to write is a guestbook!
The law of the median use case:

If you are doing something similar to what 10,000 people have already accomplished, then it is highly likely that you will succeed. If you are doing something that very few people have accomplished before, you are likely to be the one to find the defects.

Reason: **social pressure**. Think of it as like a footpath in the forest:

Paths that have been taken frequently by others are easier to follow.
Myth #4: If I have a great idea, I can become the next Bill Gates/Steve Jobs/Larry Ellison/… Reality: It is about as likely as winning the lottery.

We're all swimming in the same river. If you have a great idea, you can rest assured that at least 1000 other people had the same idea. The difference is whether you do something about it, and whether you're first to market with the innovation.

From a friend who is a psychiatrist: "I won't go hungry if your students have that attitude."
Social guidelines for paradigm adoption

1. Consider the **downsides** of adoption: there is nothing more difficult than **swimming against a social river**.
2. Consider success and failure stories as **evidence**, and not as **truth**.
3. Remember the differences between **pattern** and **ritual**.
4. Seek to function via **chemistry** rather than **alchemy**.
5. Use the **law of the median** to your advantage, rather than to your disadvantage: avoid **superfluous divergence** from the median use case.
Case history: .NET versus LAMP

Problem: SOA programming
Two service stacks to build.
Relatively inexperienced but motivated programmers.
Paradigms: .NET, LAMP
Attributes:
Windows-based.
Homogeneous environment: only on Windows. Mono is problematic.
One development environment: Visual Studio

Pluses:
High adoption rate: high state of knowledge.
Business pressure to make development seamless.
Pre-existing patterns for SOA programming.
Patterns well documented and seamlessly deployable.

Minuses:
Single-vendor environment: cost of deployment is high.
Relatively inflexible: if you're doing SOAP, you're fine; if you're doing REST, prepare to swim upstream.
LAMP: Linux/Apache/Mysql/PHP (or Perl)
    Freely available.
    Free platform.
    Widely adopted.

Pluses:
    Free development environment.
    Lots of social pressure and well-tested applications and modules.

Minuses:
    No social pressure toward high integration.
    Minimal support for specific web paradigms, e.g., SOA.
    Guild knowledge needed to program many kinds of functions.
### A strategic comparison of .NET and LAMP

Wednesday, November 14, 2012
3:44 PM

<table>
<thead>
<tr>
<th>.NET</th>
<th>LAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seamless deployment of SOA</td>
<td>Deployment requires much knowledge</td>
</tr>
<tr>
<td>Deployments cost money</td>
<td>Deployment is free.</td>
</tr>
<tr>
<td>Relatively inflexible SOA: just SOAP</td>
<td>SOAP and REST support.</td>
</tr>
<tr>
<td>Startup knowledge: low</td>
<td>Startup knowledge: high</td>
</tr>
</tbody>
</table>

**Decision:** .NET

**Subtleties:**
- Capabilities of community were primary motivating factor.
- Cost of deployment was trumped by time to market.
To continue our story,

Now we need to deploy the services in the cloud.
Two choices: Azure and Amazon
Azure

Seamless deployment of existing .NET applications
Hopelessly tied to MSSQL.
High operating cost.
Platform as a service: you don't load Windows.
Amazon

Tied to LAMP
Lower cost to operate.
Microsoft stacks cost extra.
Infrastructure as a service: you load your own OS.
<table>
<thead>
<tr>
<th>Amazon</th>
<th>Azure</th>
</tr>
</thead>
<tbody>
<tr>
<td>$.10/GB/Month</td>
<td>$.15/GB/Month</td>
</tr>
<tr>
<td>No distributed SQL solution</td>
<td>Distributed SQL supports scalable search, costs $5/Gb/Month!</td>
</tr>
<tr>
<td>Cost of adoption: high</td>
<td>Cost of adoption near zero</td>
</tr>
<tr>
<td>New expertise base required</td>
<td>Existing expertise base sufficient.</td>
</tr>
<tr>
<td>High-risk port.</td>
<td>Risk near zero.</td>
</tr>
</tbody>
</table>

Decision: Azure.
Why: high cost of operating is offset by very low risk and immediate deployment. Cost of operations motivates development of LAMP version **after** Azure is in place.
A story of social pressure

I was a first adopter of the Foundry ServerIron product. This is a high-end dynamic load balancer.

I found a really serious defect: NFS breaks because the switch fails to forward UDP fragmented packets.

They fixed it.

I validated the fix.

A month later, the new version arrived.

It was broken again. Same way.

I go through two more iterations.

#2: They fix it, send it to me, put it into the main branch, promise no more problems.

#3: I get another release: broken.

Reason: programmer error. No documentation of "why?"

Finally, it gets into the release permanently.