Compared with the time we spend in development,
our systems spend much longer in production.

Surely it's worth us spending a little of our development time to make our systems easier to manage.

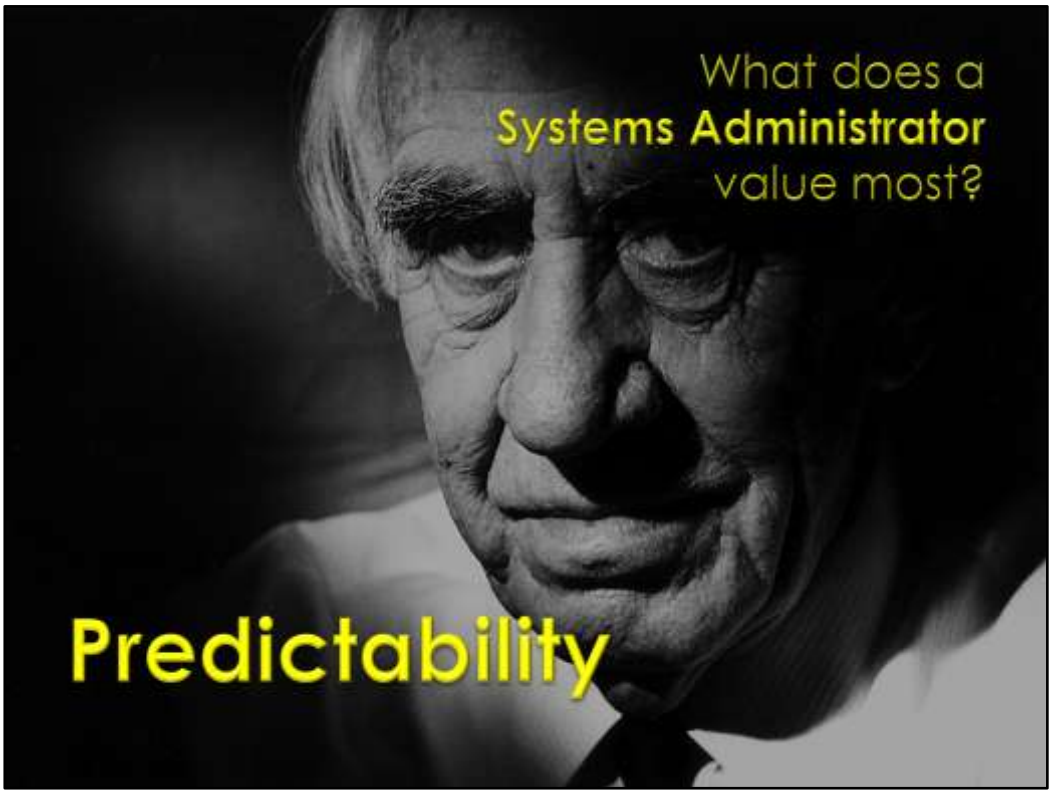
Making use of some strong stereotypes
Intended to make a point
Not reflective of any one I know

Definition

A developer is someone who works on the construction of a system

What does a Developer value most?

Working Code.
Readable Code.
Elegant Code.
Clear Code.
Performant Code.
Simple Code.
Effective Code.
Testable Code.
Maintainable Code.
Efficient Code.
Tested Code.



Definition
A systems administrator is someone who is responsible for looking after a production system

What does a Systems Administrator value most?
Predictability

Systems Administrators are charged with keeping things going.

Anything that gets in the way of that goal is bad.

CHANGE is good. You go first. (Richard Campbell, Dot Net Rocks)

Developers work on one system at a time
Systems Administrators work on many systems all the time

Developers know (almost) everything about one system
Systems Administrators know something about every system

Developers spend weeks or months (or longer) on one system
Systems Administrators spend hours (or minutes) on each system

Developers have to fix problems before the end of the sprint/phase/project
Systems Adminstrators have to fix problems by the end of the day

The key question ...

The key question ...

There is some **bad news** ...

There is some bad news ...
... You do need to write documentation

START EARLY
don't leave it to the end.
Start early in the project and capture information as it occurs to you.

UPDATE OFTEN
Find an approach that works for you/your team.
Something lightweight so you can update things easily and quickly;
Probably not Word.
OneNote/Wiki. Even Email

SUGGESTION
Here's a structure that works
Architecture – a summary of physical deployment
Symptoms – a list of failure modes that users might encounter
Actions – possible remedies
Procedures – step by step instructions for common procedures
Reference – additional reference material

PRACTICE
Nothing should be done for the first time **at go-live.**
Everything should be practiced – rehearsed – before it's needed for real.

SIMPLIFY
Regular rehearsal will reveal where the pain points are in your process
Attend to them! Work to reduce the pain.

AUTOMATE
Avoid the "simple 48 step checklist"

TIPS
What *can* be automated, *should* be automated.
What *can* be eliminated *should* be removed.

TECHNOLOGIES
MSI installers – Batch/Cmd files – Powershell – NAnt – MSBuild

SUGGESTION
Deployment to ALL your test and staging environments should be done the same way as go-live.

SECURITY
Proper controls of access are necessary.
Security must not be an afterthought

Sufficient security has to be designed in from the start, because you can't design it in later
Note that you don't have to build it, necessarily, but you do have to know what you're going to do

CONSIDERATIONS
Authentication – how will you know WHO a particular user is?
Authorisation – how will you know WHAT they are allowed to do?
Disclosure – you will you LIMIT what they can see?

THE CLEAR BOX
When we are developing, the system is a clear box – we can take the lid off to see what is going on when something goes wrong.

THE BLACK BOX
When the system is in production, the system is a black box – administrators don't have the option of taking off the lid.
We must provide a way to look inside the system.

TRACING
Leaving a trail allowing someone to trace program flow from layer to layer
Or even object to object or class to class
High volume – for developers

LOGGING
Recording the results of actions
Low volume – for administrators

CONTEXT
Don't just log the failure message
Conceals critical information that may be necessary to solve the problem
Also log the context of the error
Date/Time/User/Server/Action/Parameters

Keep things as simple as they can be, but no simpler.

ESSENTIAL COMPLEXITY
Some complexity is inherent to the problem you're trying to solve.
If you leave out some part of this, you can't solve the problem.

ACCIDENTAL COMPLEXITY
Some complexity comes about from the way that you try to solve the problem.
Work to reduce/eliminate accidental complexity as far as you can

SOURCES OF COMPLEXITY
Business Domain – Rules, interactions
Performance – scaling up, scaling out
Technology choices – choice


Towards Supportability
Five
Guidelines
Three
Actions

DOCUMENTATION
Start working on your release documentation today.
Capture the information as it occurs to you, don't leave it to the end.

TECHNOLOGIES
OneNote – open all the time to capture
Wiki – web page open all the time
Word – document open all the time
Reduce friction

COMMUNICATION
Start opening the paths of communication with your Systems Administrators
Work to reduce the surprise factor
Treat *System Administrators* as another important group of system users
Get some Admin involvement, even kicking and screaming

TREAT THE PAIN
Start work to reduce the pain of deployment and support

What is pain?
Every time you have to crack the lid to find out why something went wrong
Every step you have to perform while building or deploying or starting
Every check you have to perform manually

Automate everything you can

Don't leave PAIN untreated.
Take steps to make things easier.

Another place to avoid the simple 48 point checklist

HISTORIC PAIN
Current Policies/Opinions/Conventions are often driven by historical pain.
Don't ignore these – you risk getting the people you depend upon offside
Find out the source of the pain – is it still valid or not? Tread carefully!

CURRENT PAIN
As you work on the system, pay attention to things that are difficult to do ... Make them easy!
You make your own life easier, as well as that of other people
Get new people to try things out – find out pain to which you're already immune

FUTURE PAIN
When you have a choice, make decisions now to make your future easier


Bevan Arps
bevan@nichesoftware.co.nz
http://www.nichesoftware.co.nz