# Ubuntu at Google

Thomas Bushnell, BSG
Google, Inc.

## Challenging user population

- Tens of thousands of employees including
  - graphic designers
  - managers
  - software engineers
  - systems engineers
  - translators
  - ...
- Including
  - People who wrote the original Unix
  - People who don't know what Unix is
  - Some of the best programmers in the world
  - Some who know next to nothing about the internals of a computer system

## Challenging demands

- Pushing workstations to their limits
- Extremely large codebases
- Very rapidly moving development cycles
- Unusual cost/benefit ratios
  - What's the cost of a reboot?
  - Custody of users' data
  - New hardware vs. cost of supporting old hardware

## What is Goobuntu?

- A light skin over standard Ubuntu like most LISA cases:
  - We don't customize UI and the like
  - Centralized administration with puppet and apt
  - LDAP-based user database
  - Automated release testing

## Goobuntu: Unusual demands

- Security requirements
  - banned packages
  - special in-house user authentication
  - pushing the state of the art in network authentication
  - extremely high-profile security target

## Goobuntu: Unusual demands

- users develop special in-house build systems for large codebases and shared development
- internal apt repository framework
- very high cost for mistakes
- diverse developers:
  - large scale perforce code bases using custom build systems
  - android and chrome using git and free software development tools
  - every corner case of UI desires and habits

## Why does Goobuntu use LTS?

Upgrading is expensive:

- hundreds of locally built packages
- even small changes are expensive
- destabilizing changes without obvious benefit
- new UI is not as exciting for our users
- very cautious adoption and phase-in process

But we lose...

- Newer versions of important stuff (e.g. KDE)
- Participation in most Ubuntu release cycles

## Canaries

- Tester pools are not sufficient
- Automated push of changes to small numbers of users
- Detection of failures with very speedy rollback

Results:

- more willingness to take beneficial risks
- less harm from buggy pushes
- less user disruption and more functional changes: profit!

## Automate yourself out of a job

- Humans do not exist to turn cranks

- Do not page a human to do a task which the system could have done: automated fault correction

- The ideal number of pages is *not zero*

- Reducing human involvement is richly rewarded

## Hope is Not a Strategy

- Computer systems fail

## Hope is Not a Strategy

- Computer systems fail
- That is, all computer systems fail

## Hope is Not a Strategy

- Computer systems fail
- That is, all computer systems fail
- Your computer systems? They fail.

## Hope is Not a Strategy

- Computer systems fail
- That is, all computer systems fail.
- Your computer systems? They fail.

Design for failure:

- System failure is not an exceptional, but an expected event
- Plan for failure of systems to be capable of being handled on a non-emergent basis
- Active monitoring is absolutely critical

## Finding cause, not placing blame

- Programmers, like systems, will make mistakes
- All programmers make mistakes.
- Your programmers? They make mistakes.

## Finding cause, not placing blame

- Programmers, like systems, will make mistakes
- Open culture around post-mortems
    - Anyone can request a post-mortems
    - What happened, and when
    - What safeguards would have helped?
    - What slowed response?
    - What would have made the people or the systems less prone to fail?
- No problem should happen the same way twice
- Human error manifests the same as system error most of the time

## Thanks and appreciation

- Support teams at Canonical
- Development teams at Canonical
- Upstream developers
- Debian developers, Ubuntu maintainers
- Fellow Googlers

## Any questions?