Cleaning code

Techniques for Large Legacy Restoration Projects

~~___~~

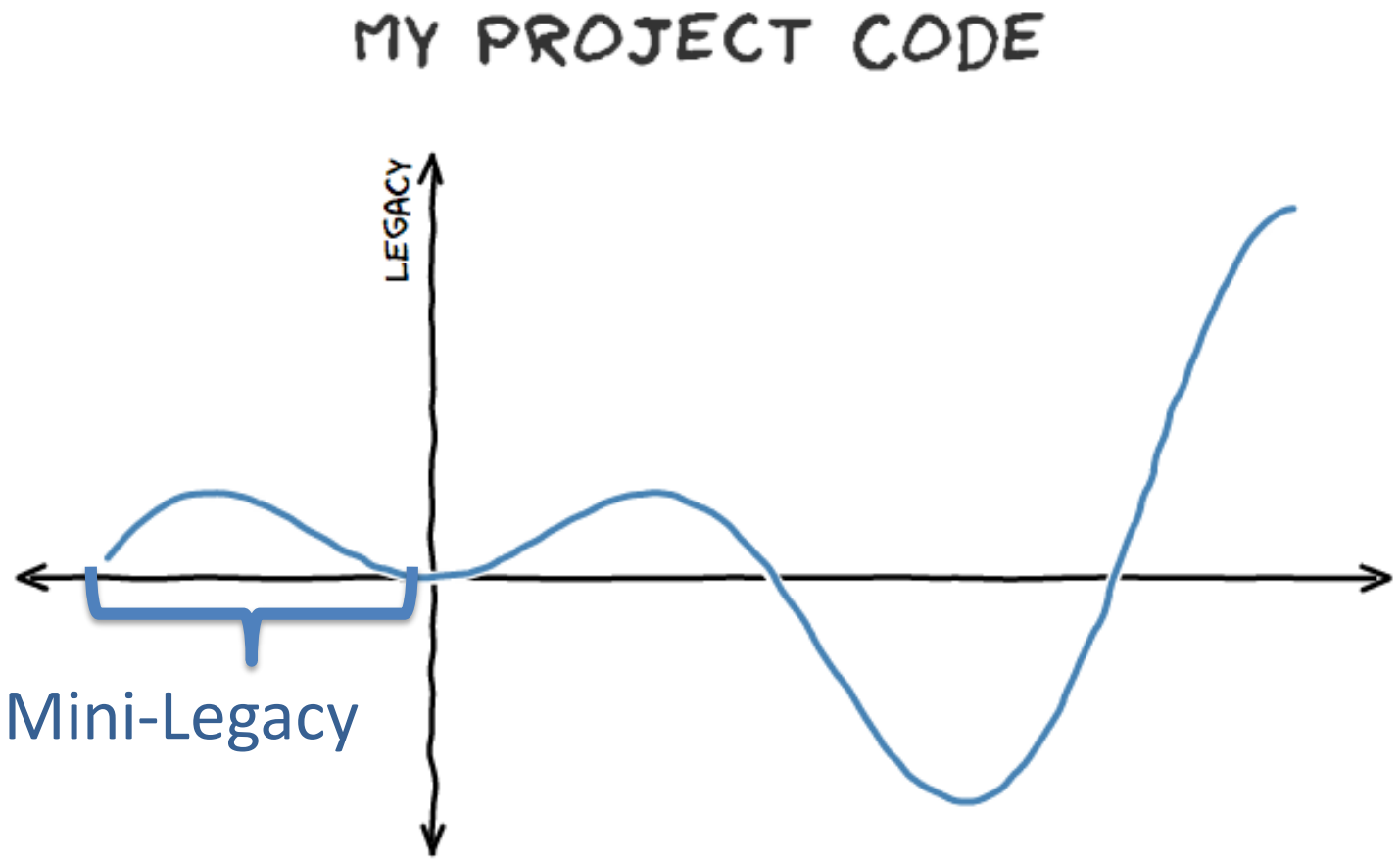
Mike Long

What's in this for you?

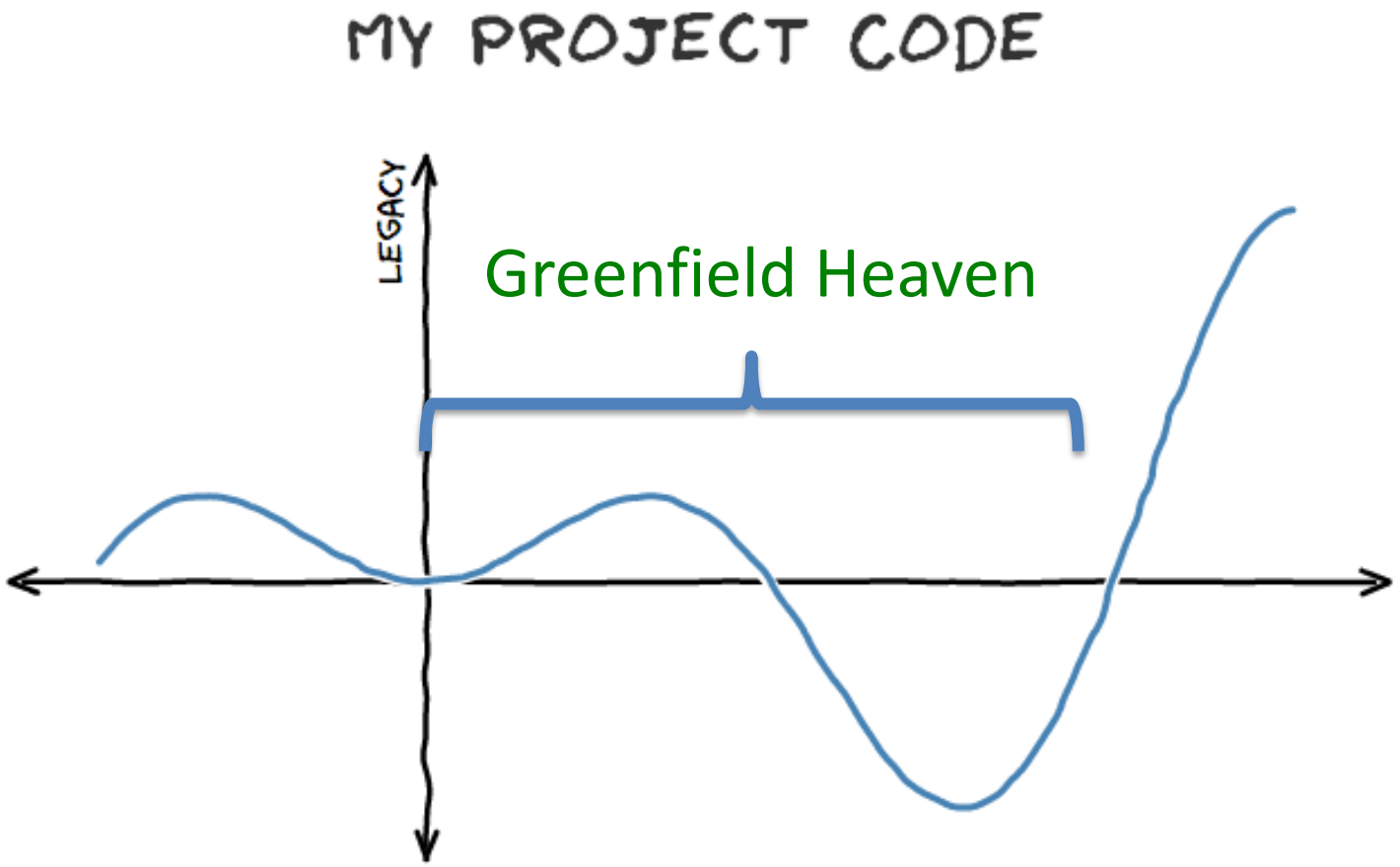
- Gain an understanding of the value of legacy
- Learn how to make the business case for remedial work in large software projects
- Know the tools necessary to be able to quantify and visualize technical debt in big projects
- How to manage large legacy restoration projects

Legacy

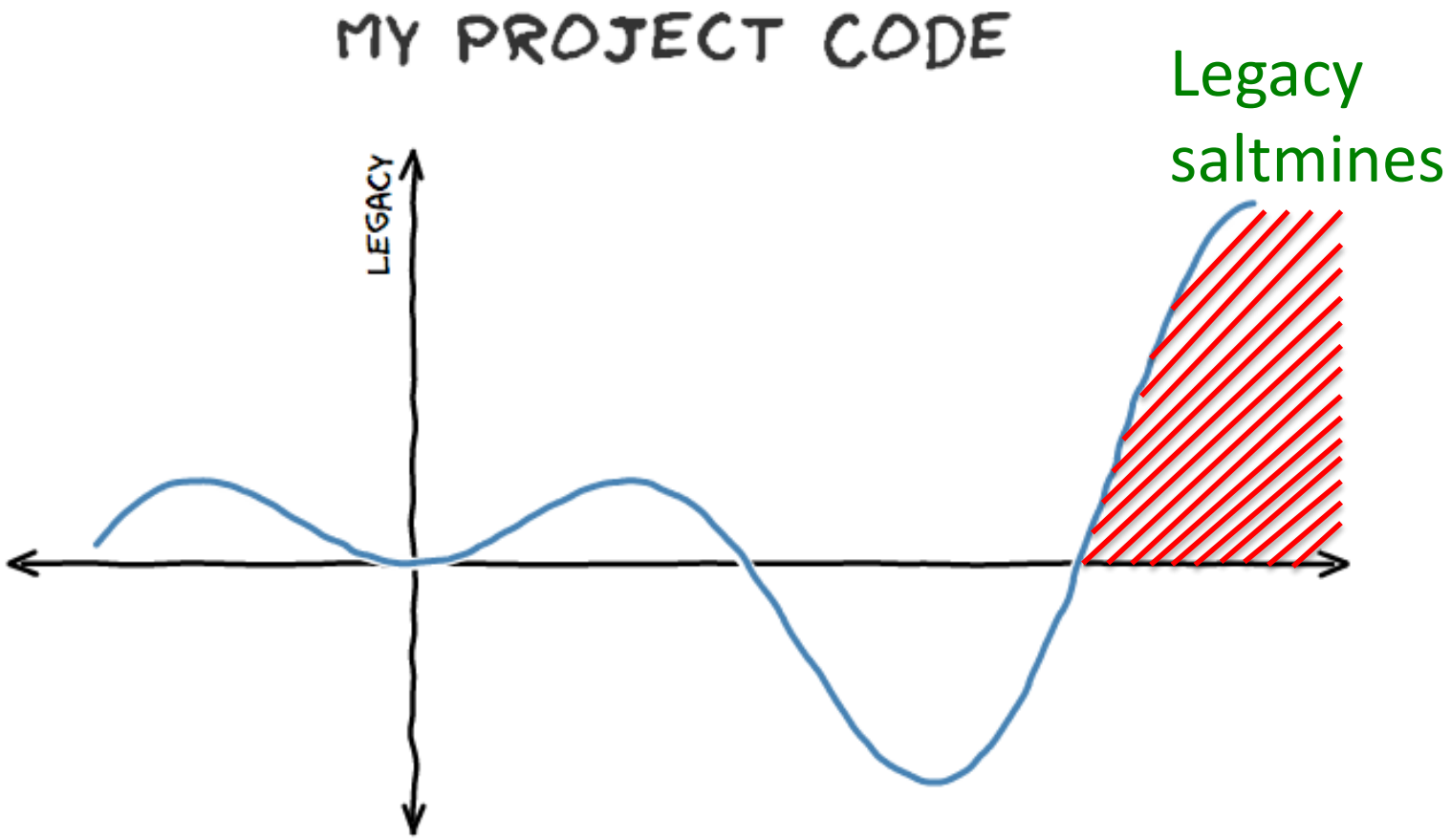
Code I have known



Code I have known



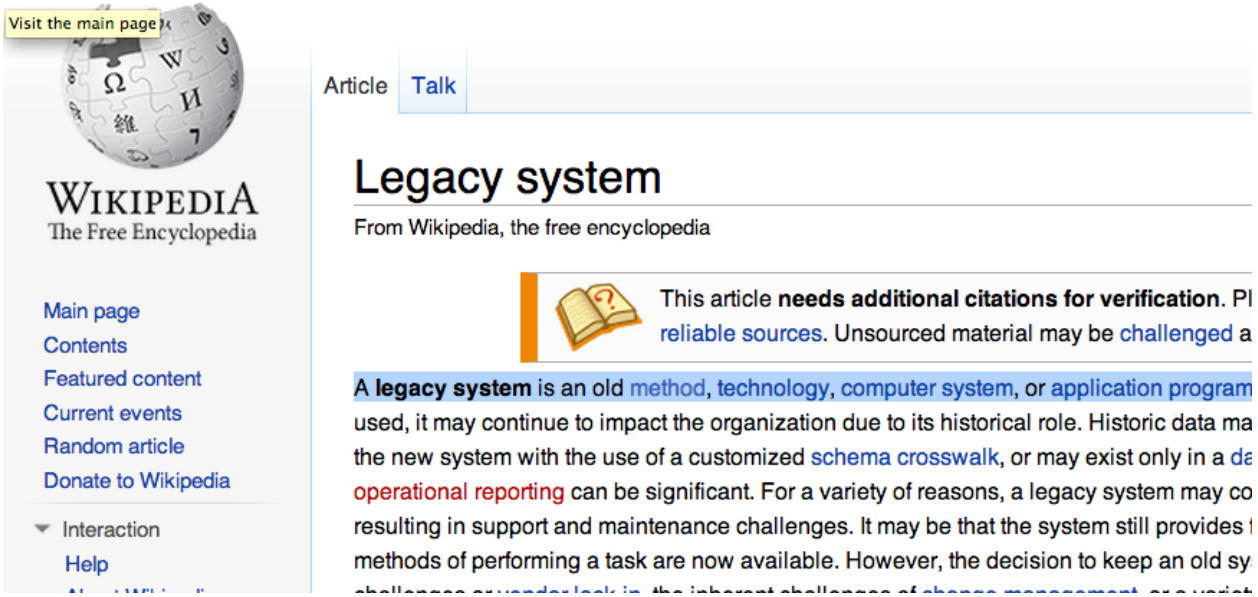
Code I have known



COBOL Rube Goldberg ~ or ~ What is a Legacy Restoration Project?

What is a **Legacy** restoration project?

"A legacy system is an old method, technology, computer system, or application program."
Wikipedia



What is a **Legacy** restoration project?

"To me, *legacy code* is simply code without tests"

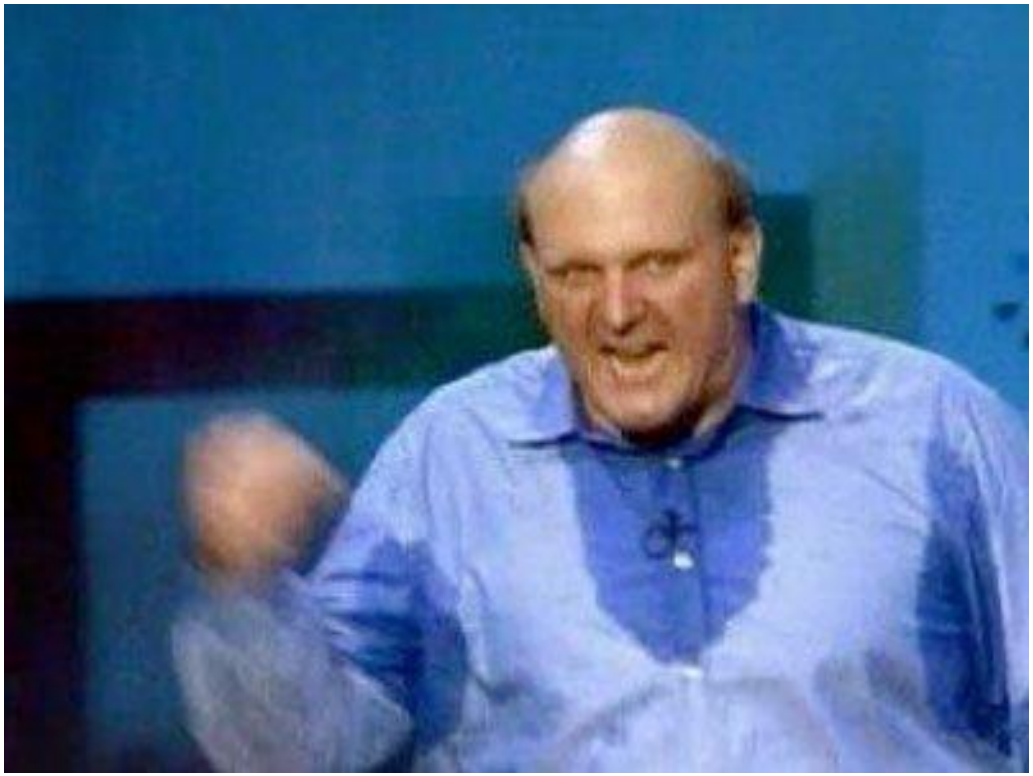
Michael Feathers

http://www.flickr.com/photos/fraserspeirs/3394782283/

What is a **Legacy** restoration project?

"Some crap made by someone else"

Developers, Developers, Developers



Large: Software at scale is a different beast

- > 1MLoC
- > 10 years old
- > 100 developers

- => Large Legacy Project

What is a Legacy **restoration project**?

- Business commitment
- Large, long lived codebase
- Valuable codebase
- Step change in quality
- Specific targets
- Time-limited

http://www.flickr.com/photos/eriginal/2908082277/

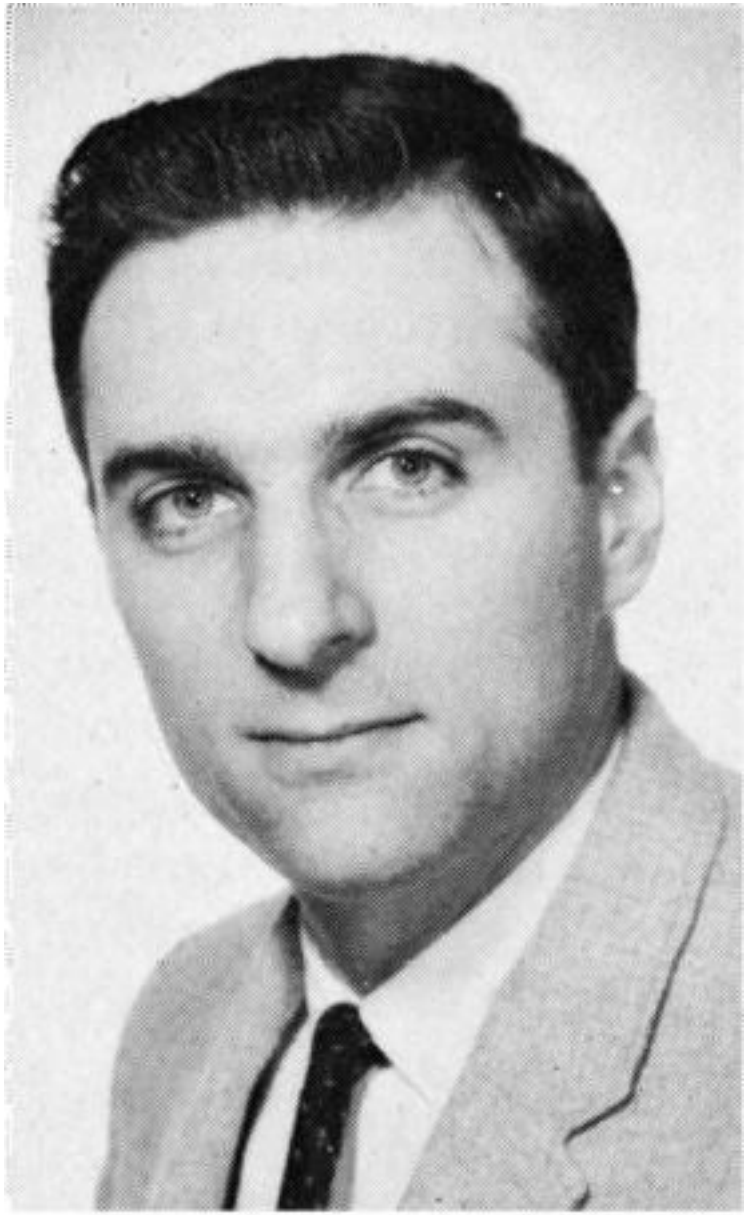
How does a codebase become a big mess?

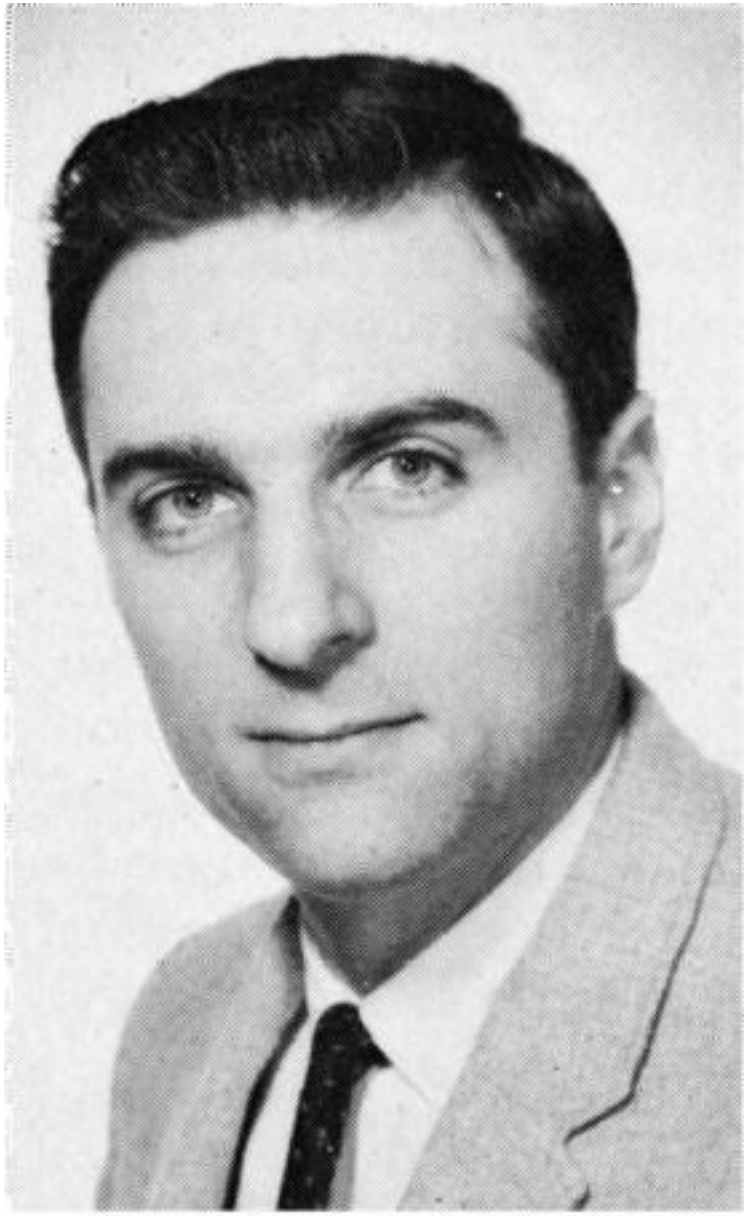
- Explosive growth – big bang development
- Sustained schedule pressure
- No quality requirements
- High staff turnover
- Success

~~How~~ Why does a codebase become a big mess?

- Feature driven development plans
- Stakeholders don’t know software
- Developers don’t know software
- Quality is tested-in, not built-in
- Under-empowered technical leaders

What to do when things get bad





HOW DO COMMITTEES INVENT?

by MELVIN E. CONWAY

That kind of intellectual activity which creates a useful whole from its diverse parts may be called the *design* of a *system*. Whether the particular activity is the creation of specifications for a major weapon system, the formation of a recommendation to meet a social challenge, or the programming of a computer, the general activity is largely the same.

Typically, the objective of a design organization is the creation and assembly of a document containing a coherently structured body of information. We may name this information the *system design*. It is typically produced for a sponsor who usually desires to carry out some activity guided by the system design. For example, a public official may wish to propose legislation to avert a recurrence of a recent disaster, so he appoints a team to explain the catastrophe. Or a manufacturer needs a new product and designates a product planning activity to specify what should be introduced.

The design organization may or may not be involved in the construction of the system it designs. Frequently, in public affairs, there are policies which discourage a group's acting upon its own recommendations, whereas, in private industry, quite the opposite situation often prevails.

It seems reasonable to suppose that the knowledge that one will have to carry out one's own recommendations or that this task will fall to others, probably affects some design choices which the individual designer is called upon to make. Most design activity requires continually making choices. Many of these choices may be more than design decisions; they may also be personal decisions the designer makes about his own future. As we shall see later, the incentives which exist in a conventional management environment can motivate choices which subvert the intent of the sponsor.[1]

stages of design

The initial stages of a design effort are concerned more with structuring of the design activity than with the system itself.[2] The full-blown design activity cannot proceed until certain preliminary milestones are passed. These include:

1. Understanding of the boundaries, both on the design activity and on the system to be designed, placed by the sponsor and by the world's realities.
2. Achievement of a preliminary notion of the system's organization so that design task groups can be meaningfully assigned.

We shall see in detail later that the very act of organizing a design team means that certain design decisions have already been made, explicitly or otherwise. Given any design team organization, there is a class of design alternatives which cannot be effectively pursued by such an organization because the necessary communication paths do not exist. Therefore, there is no such thing as a design group which is both organized and unbiased.

Once the organization of the design team is chosen, it is possible to delegate activities to the subgroups of the organization. Every time a delegation is made and somebody's scope of inquiry is narrowed, the class of design alternatives which can be effectively pursued is also narrowed.

Once scopes of activity are defined, a coordination problem is created. Coordination among task groups, although it appears to lower the productivity of the individual in the small group, provides the only possibility that the separate task groups will be able to consolidate their efforts into a unified system design.

Thus the life cycle of a system design effort proceeds through the following general stages:

1. Drawing of boundaries according to the ground rules.
2. Choice of a preliminary system concept.
3. Organization of the design activity and delegation of tasks according to that concept.
4. Coordination among delegated tasks.
5. Consolidation of subdesigns into a single design.

It is possible that a given design activity will not proceed straight through this list. It might conceivably reorganize upon discovery of a new, and obviously superior, design concept; but such an appearance of uncertainty is unflattering, and the very act of voluntarily abandoning a creation is painful and expensive. Of course, from the

[1] A related, but much more comprehensive discussion of the behavior of system-designing organizations is found in John Kenneth Galbraith's, *The New Industrial State* (Boston, Houghton Mifflin, 1967). See especially Chapter VI, "The Technostructure."

[2] For a discussion of the problems which may arise when the design activity takes the form of a project in a functional environment, see C. J. Middleton, "How to Set Up a Project Organization," *Harvard Business Review*, March-April, 1967, p. 73.

Dr. Conway is manager, peripheral systems research, at Sperry Rand's Univac Div., where he is working on recognition of continuous speech. He has previously been a research associate at Case Western Reserve Univ., and a software consultant. He has an MS in physics from CalTech and a PhD in math from Case.

“organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations.”

HOW DO COMMITTEES INVENT?

by MELVIN E. CONWAY

That kind of intellectual activity which creates a useful whole from its diverse parts may be called the *design* of a *system*. Whether the particular activity is the creation of specifications for a major weapon system, the formation of a recommendation to meet a social challenge, or the programming of a computer, the general activity is largely the same.

Typically, the objective of a design organization is the [illegible] It usually begins with a sponsor who usually desires to carry out some activity guided by the system design. For example, a public official may wish to propose legislation to avert a recurrence of a [illegible] and initiates a product planning activity to specify what should be introduced.

The design organization may or may not be involved in [illegible] industry, quite the opposite situation often prevails.

It seems reasonable to suppose that the knowledge that one will have to carry out one's own recommendations or [illegible] probably affects some [illegible] is called upon to make. Most design activity requires continually making choices. Many of these choices may be more than design decisions; they may also be personal decisions the designer makes about his own future. As we shall see later, the incentives which exist in a conventional management environment can motivate choices which subvert the intent of the sponsor.[1]

stages of design

The initial stages of a design effort are concerned more with structuring of the design activity than with the system itself.[2] The full-blown design activity cannot proceed until certain preliminary milestones are passed. These include:

1. Understanding of the boundaries, both on the design activity and on the system to be designed, placed by the sponsor and by the world's realities.
2. Achievement of a preliminary notion of the system's organization so that design task groups can be meaningfully assigned.

We shall see in detail later that the very act of organizing a design team means that certain design decisions have already been made, explicitly or otherwise. Given any design team organization, there is a class of design alternatives which cannot be effectively pursued by such an organization because the necessary communication paths do not exist. Therefore, there is no such thing as a design group which is both organized and unbiased.

Once the organization of the design team is chosen, it is possible to delegate activities to the subgroups of the [illegible]ization. Every [illegible]ation is made and some[illegible] scope of [illegible]ved, the class of design [illegible] which [illegible] effectively pursued is also narrowed.

Once scopes of activity are defined, a coordination prob[illegible] is created. Coordination among task groups, although [illegible] of the individual in the [illegible] possibility that the separate task groups will be able to consolidate their efforts into a unified system design.

Thus the life cycle of a system design effort proceeds [illegible] according to the ground [illegible]:

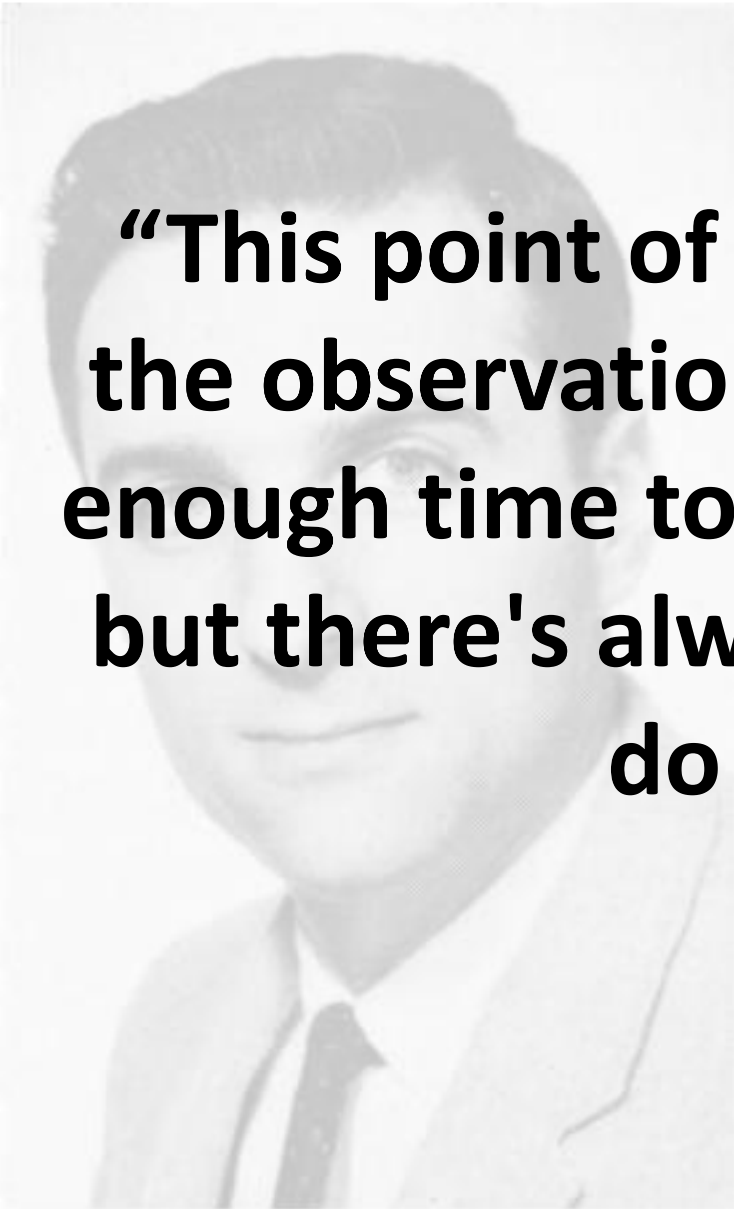
2. Choice of a preliminary system concept.
3. Organization of the design activity and delegation of tasks according to that concept.
4. Coordination among delegated tasks.
5. Consolidation of subdesigns into a single design.

It is possible that a given design activity will not proceed straight through this list. It might conceivably reorganize upon discovery of a new, and obviously superior, design concept; but such an appearance of uncertainty is unflattering, and the very act of voluntarily abandoning a creation is painful and expensive. Of course, from the

[1] A related, but much more comprehensive discussion of the behavior of system-designing organizations is found in John Kenneth Galbraith's, *The New Industrial State* (Boston, Houghton Mifflin, 1967). See especially Chapter VI, "The Technostructure."

[2] For a discussion of the problems which may arise when the design activity takes the form of a project in a functional environment, see C. J. Middleton, "How to Set Up a Project Organization," *Harvard Business Review*, March-April, 1967, p. 73.

Dr. Conway is manager, peripheral systems research, at Sperry Rand's Univac Div., where he is working on recognition of continuous speech. He has previously been a research associate at Case Western Reserve Univ., and a software consultant. He has an MS in physics from CalTech and a PhD in math from Case.



"This point of view has produced the observation that there's never enough time to do something right, but there's always enough time to do it over"


Bankruptcy Ct
Counseling Wy

Rewrite trap #1: re-implementing existing features == commercial suicide

- Netscape
- Borland
- Microsoft

Joel on Software

Things You Should Never Do, Part I

by Joel Spolsky

Thursday, April 06, 2000

Netscape 6.0 is finally going into its first public beta. There never was a version 5.0. The last major release, version 4.0, was released almost three years ago. Three years is an *awfully* long time in the Internet world. During this time, Netscape sat by, helplessly, as their market share plummeted.

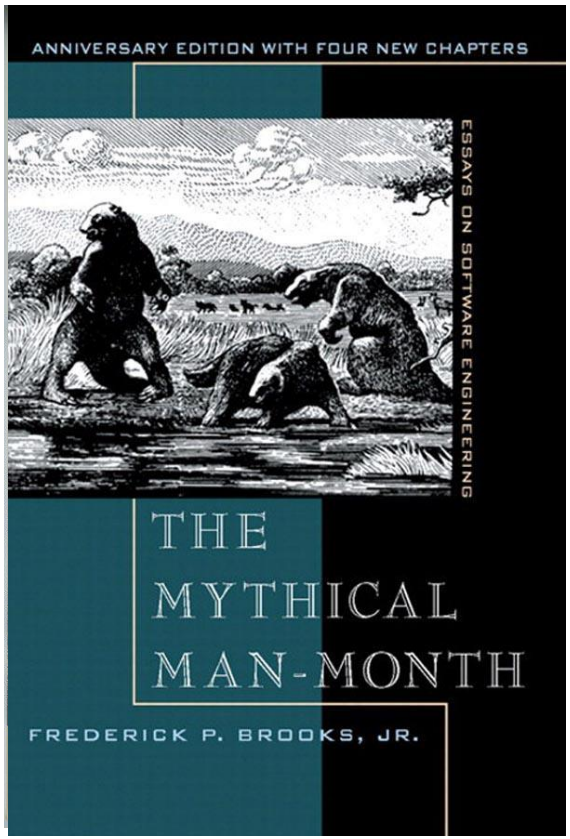
It's a bit smarmy of me to criticize them for waiting so long between releases. They didn't do it *on purpose*, now, did they?

Well, yes. They did. They did it by making the **single worst strategic mistake** that any software company can make:

They decided to rewrite the code from scratch.

Netscape wasn't the first

Rewrite trap #2: The 2nd System Effect



"This second is the most dangerous system a man ever designs…The result, as Ovid says, is a "big pile."

Too big to fail?

Should we declare bankruptcy?

For a re-write to be worth it:

- We really have "*enough time to do it over*"
  - *Money is no object, re-implementation of existing features*
- We use incremental value delivery to stave off the second systems effect
  - mature and disciplined team
- We can mitigate the knowledge loss
  - small and simple system, *and* the same team as initial implementation
- And the existing platform is facing obsolescence

For all other cases, rewrite is commercial suicide

Tools and techniques

Identify & remove waste

The Carrying Cost of Code

Ted Dziuba
@dozba

Following

"Code is a liability, functionality is an asset."

Reply Retweet Favorite More

12 RETWEETS

9 FAVORITES

4:17 AM - Jun 7, 2010

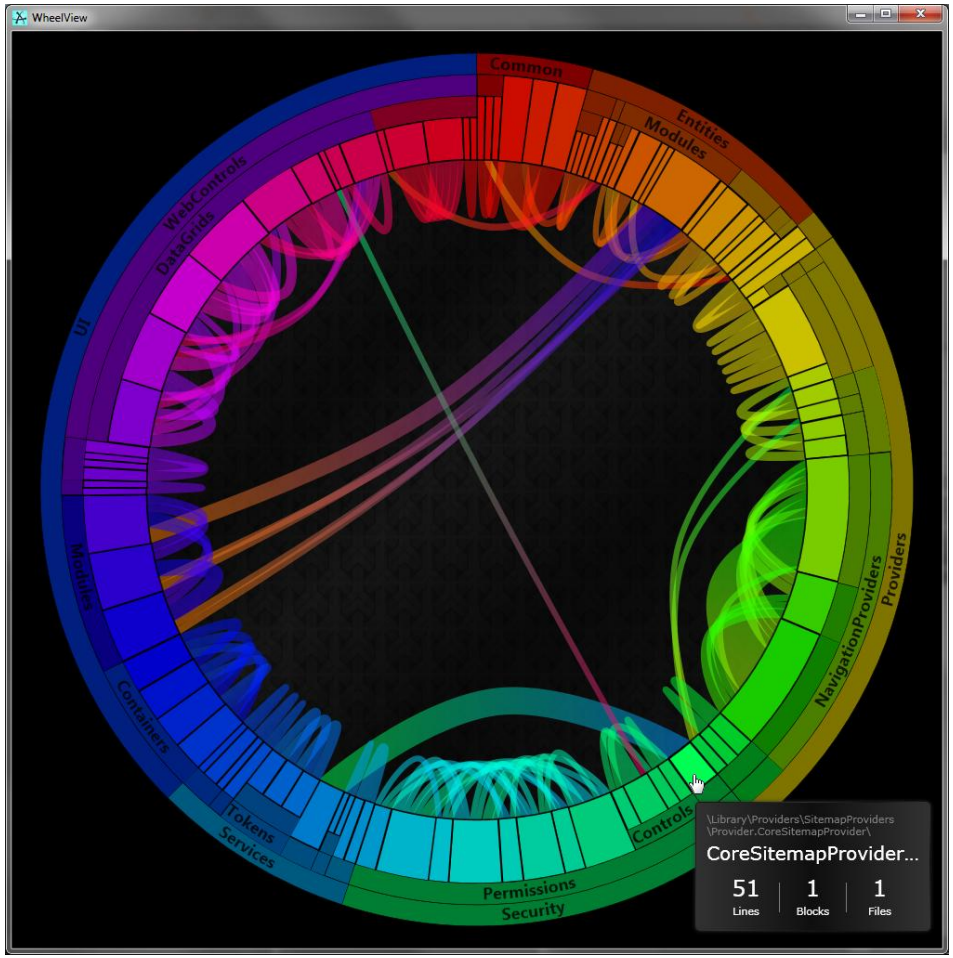
Carrying Costs

- Large projects take time to build
- Defect tracking systems
- Communication costs
- Knowledge costs

How much of your code is dead?

- Callcatcher
- Listen to your linker
  - [Obsolete]
  - __declspec(deprecated(“**TESTING DEAD**”))

Duplicate code



Tests are inventory

- Tests are inventory, whether scripts, unit tests, or gui tests
- Long term failing tests
  - Delete them or fix them,
  - then automate this process
- Flicker tests

There are no fixed costs

- Exploit the huge opportunities for waste reduction in large legacy
  - If you have a big development population, any waste reduction is a huge boost to productivity
  - Think build times, feedback delays, broken builds, testing cycles, installation times
- But baseline and monitor it!
- Measure it in business terms ($$$)

Pete Goodliffe
@petegoodliffe

Follow

It's remarkable what can be achieved with blunt tools and perspiration.

11:29 AM - 05 Mar 12

1 RETWEET **1** FAVORITE

Sharp tools

- Look for waste introducing systems
  - Legacy Version control systems
  - Legacy Defect tracking
- Introduce productivity enhancing tools
  - Continuous Integration
  - Automation

Automate the donkey work

- **Builds**
- **Dev Setup**
- **Deployment**
- **Testing**
- **Support**

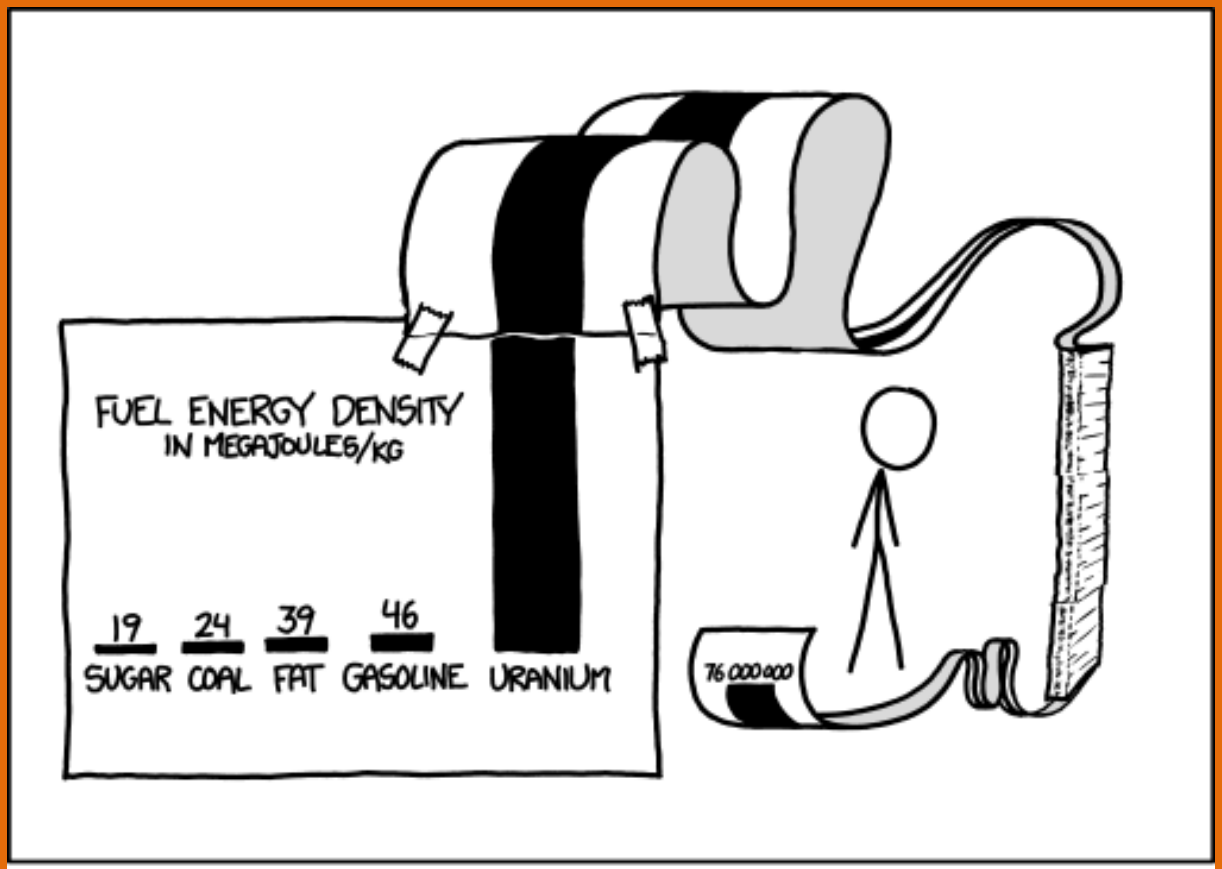
Churn

- What is changing?
- What is not changing?
  - The active set of classes
- Drive your decisions on where to put your effort

Static Analysis

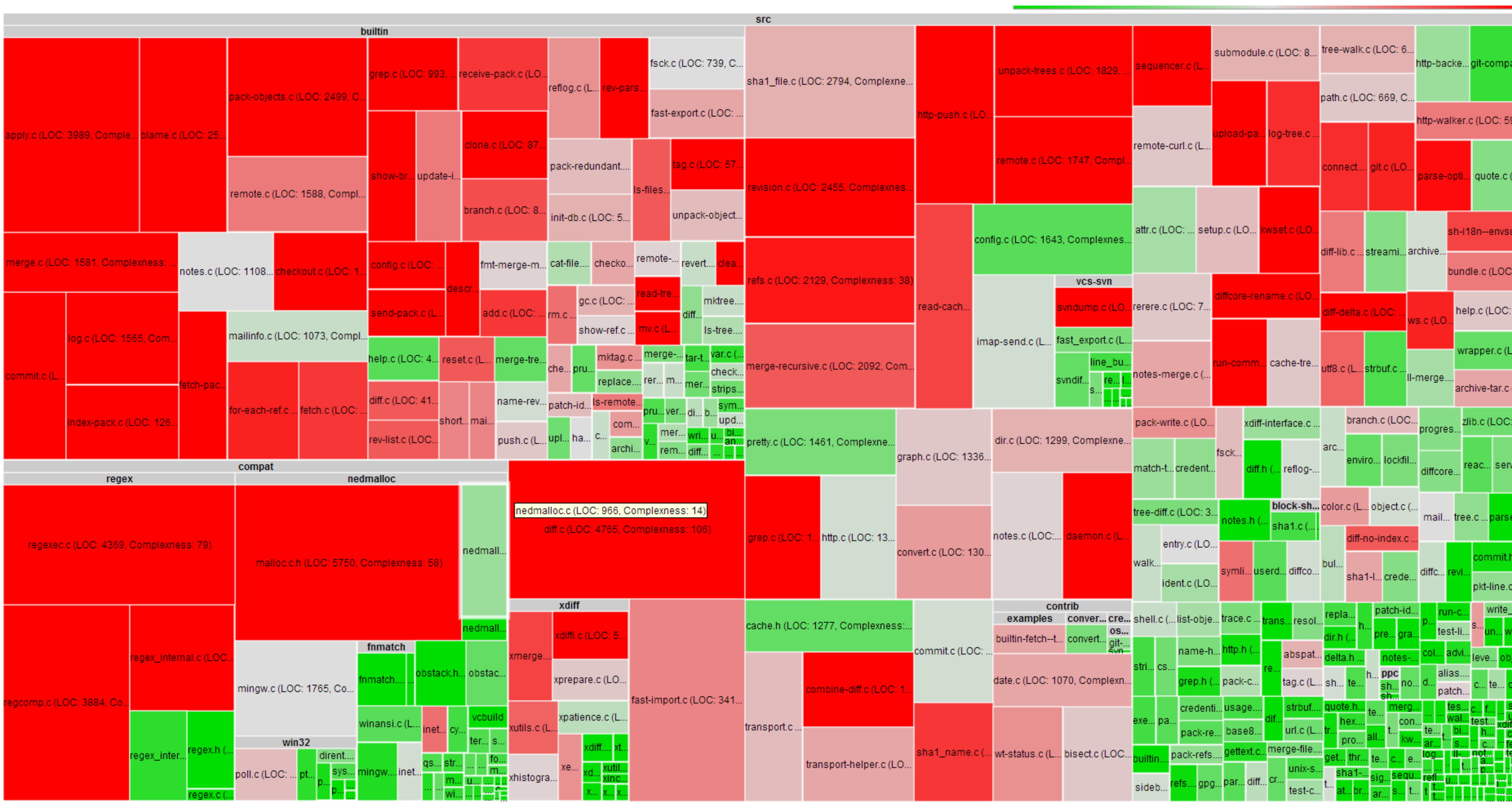
- The compiler is the first port of call
  - -wall or /warn:4
- I have had very good experiences with coverity and cppcheck, YMMV
- Resource leak detection and memory corruption detection

Visualization



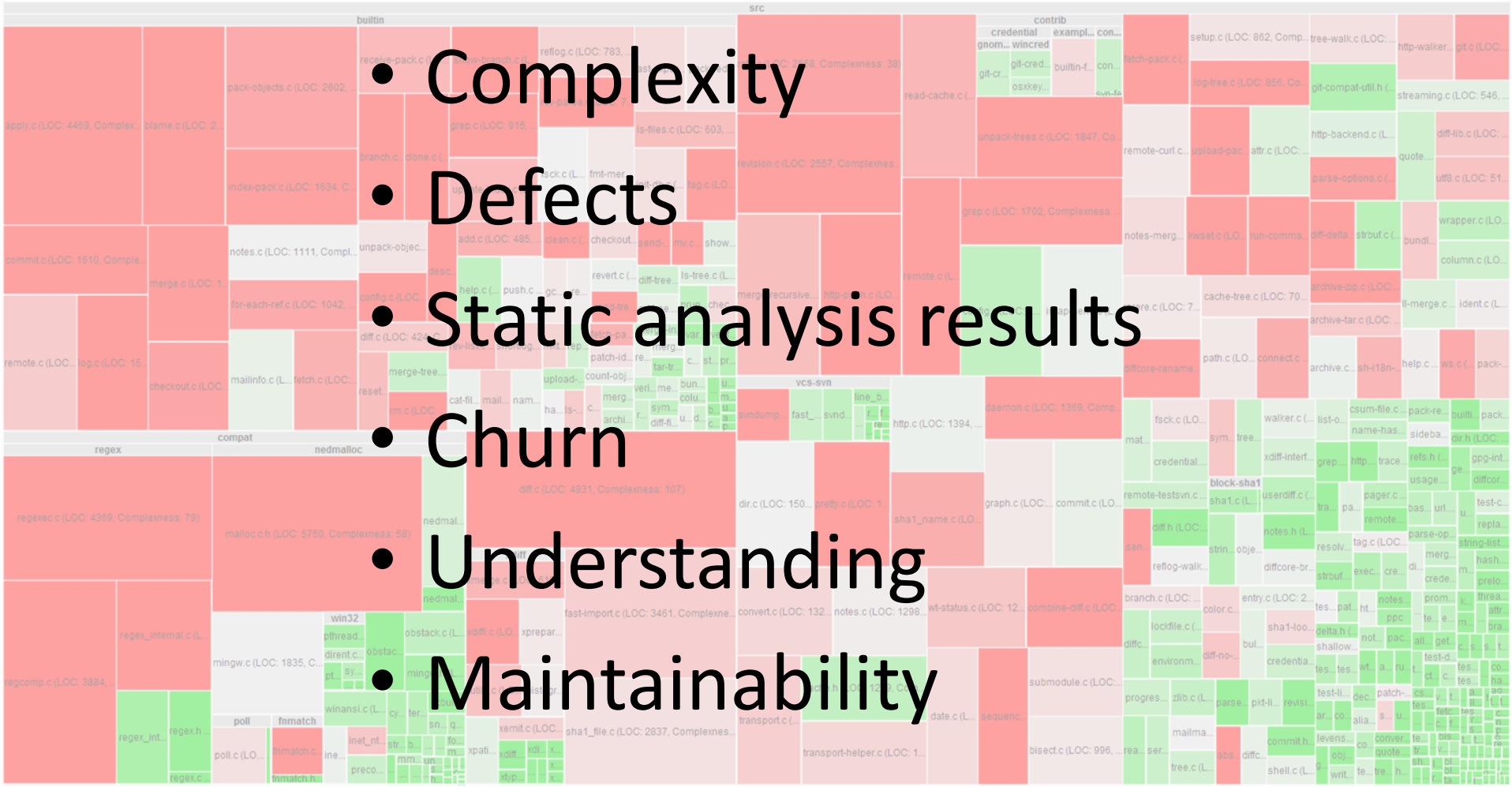
SCIENCE TIP: LOG SCALES ARE FOR QUITTERS WHO CAN'T FIND ENOUGH PAPER TO MAKE THEIR POINT *PROPERLY.*


Visualization


src
builtin
apply.c (LOC: 3989, Comple...
blame.c (LOC: 25...
pack-objects.c (LOC: 2499, C...
remote.c (LOC: 1588, Compl...
grep.c (LOC: 993, ...
receive-pack.c (LO...
reflog.c (L...
rev-pars...
fsck.c (LOC: 739, C...
fast-export.c (LOC: ...
clone.c (LOC: 87...
branch.c (LOC: 8...
pack-redundant...
init-db.c (LOC: 5...
tag.c (LOC: 57...
unpack-object...
ls-files...
merge.c (LOC: 1581, Complexness: ...
notes.c (LOC: 1108...
checkout.c (LOC: 1...
config.c (LOC: ...
fmt-merge-m...
cat-file...
send-pack.c (L...
add.c (LOC: ...
log.c (LOC: 1565, Com...
commit.c (L...
index-pack.c (LOC: 126...
fetch-pac...
mailinfo.c (LOC: 1073, Compl...
for-each-ref.c ...
fetch.c (LOC: ...
help.c (LOC: 4...
reset.c (L...
merge-tre...
diff.c (LOC: 41...
rev-list.c (LOC...
name-rev...
push.c (L...
mktag.c ...
replace...
compat
regex
nedmalloc
regexec.c (LOC: 4369, Complexness: 79)
malloc.c.h (LOC: 5750, Complexness: 58)
nedmall...
nedmalloc.c (LOC: 966, Complexness: 14)
diff.c (LOC: 4765, Complexness: 106)
regcomp.c (LOC: 3884, Co...
regex_internal.c (LOC...
regex_inter...
regex.h (...
mingw.c (LOC: 1765, Co...
fnmatch
fnmatch...
obstack.h...
win32
poll.c (LOC: ...
winansi.c (L...
xdiff
xmerge...
xdiffi.c (LOC: 5...
xprepare.c (LO...
xutils.c (L...
xpatience.c (L...
xhistogra...
fast-import.c (LOC: 341...
sha1_file.c (LOC: 2794, Complexne...
revision.c (LOC: 2455, Complexnes...
refs.c (LOC: 2129, Complexness: 38)
merge-recursive.c (LOC: 2092, Com...
pretty.c (LOC: 1461, Complexne...
grep.c (LOC: 1...
http.c (LOC: 13...
cache.h (LOC: 1277, Complexness:...
transport.c ...
combine-diff.c (LOC: 1...
transport-helper.c (LO...
http-push.c (LO...
read-cach...
unpack-trees.c (LOC: 1829, ...
remote.c (LOC: 1747, Compl...
config.c (LOC: 1643, Complexnes...
imap-send.c (L...
vcs-svn
svndump.c (LO...
fast_export.c (L...
graph.c (LOC: 1336...
convert.c (LOC: 130...
commit.c (LOC: ...
sha1_name.c (...
dir.c (LOC: 1299, Complexne...
notes.c (LOC:...
daemon.c (L...
contrib
examples
builtin-fetch--t...
date.c (LOC: 1070, Complexn...
wt-status.c (L...
bisect.c (LOC...
sequencer.c (L...
submodule.c (LOC: 8...
tree-walk.c (LOC: 6...
path.c (LOC: 669, C...
http-walker.c (LOC: 59...
remote-curl.c (L...
upload-pa...
log-tree.c ...
attr.c (LOC: ...
setup.c (LO...
kwset.c (LO...
diffcore-rename.c (LO...
rerere.c (LOC: 7...
notes-merge.c (...
run-comm...
cache-tre...
pack-write.c (LO...
xdiff-interface.c ...
tree-diff.c (LOC: 3...
bundle.c (LOC:...
help.c (LOC: ...
wrapper.c (L...
archive-tar.c (...
zlib.c (LOC: ...
diff-no-index.c ...
commit.h...
pkt-line.c ...

src
builtin
apply.c (LOC: 4469, Complex...
blame.c (LOC: 2...
pack-objects.c (LOC: 2602, ...
index-pack.c (LOC: 1634, C...
receive-pack.c (LO...
show-branch.c (L...
reflog.c (LOC: 783, ...
rev-parse.c (LOC: 7...
fast-expor...
pack-red...
ls-files.c (LOC: 603, ...
grep.c (LOC: 915, ...
branch.c...
clone.c (...
update-index.c (L...
fsck.c (L...
fmt-mer...
init-db.c (...
tag.c (LO...
commit.c (LOC: 1610, Comple...
merge.c (LOC: 1...
notes.c (LOC: 1111, Compl...
for-each-ref.c (LOC: 1042, ...
unpack-objec...
desc...
config.c (LOC...
add.c (LOC: 485, ...
clean.c (...
checkout...
send-...
mv.c...
show...
help.c (...
push.c ...
gc....
re...
revert.c (...
diff-tree...
ls-tree.c (L...
read-tre...
mktree...
prun...
chec...
remote.c (LOC...
log.c (LOC: 15...
checkout.c (LOC...
mailinfo.c (L...
fetch.c (LOC:...
diff.c (LOC: 424, C...
merge-tree...
reset...
rm.c (LOC:...
rev-list.c ...
shortlog...
mkt...
rep...
fetch-pa...
patch-id...
merge-in...
merg...
tar-tr...
var...
verif...
cat-fil...
mail...
nam...
upload-...
count-obj...
merg...
archi...
ha...
ls-...
veri...
me...
sym...
diff-fi...
bun...
colu...
compat
regex
nedmalloc
regexec.c (LOC: 4369, Complexness: 79)
regcomp.c (LOC: 3884, ...
regex_internal.c (L...
regex_int...
regex.h ...
regex.c ...
malloc.c.h (LOC: 5750, Complexness: 58)
nedmal...
nedmal...
mingw.c (LOC: 1835, C...
win32
pthread...
dirent.c...
obstac...
obstack.c (L...
mingw.h (L...
winansi.c (L...
vcbuild
poll
fnmatch
poll.c (LO...
fnmatch.c...
fnmatch.h...
inet_nt...
preco...
diff.c (LOC: 4931, Complexness: 107)
xdiff
xmerge.c (LOC: 619,...
xdiff.c (LO...
xprepar...
xutils.c (LO...
xhistogr...
xemit.c (LOC...
xpati...
xdiff...
xtyp...
fast-import.c (LOC: 3461, Complexne...
sha1_file.c (LOC: 2837, Complexnes...
refs.c (LOC: 2558, Complexness: 38)
revision.c (LOC: 2557, Complexnes...
merge-recursive...
http-push.c (LO...
read-cache.c (...
remote.c (L...
contrib
credential
exampl...
con...
gnom...
wincred
git-cr...
git-cred...
osxkey...
builtin-f...
unpack-trees.c (LOC: 1847, Co...
grep.c (LOC: 1702, Complexness: ...
config.c (LOC: 1...
imap-send.c (L...
vcs-svn
svndump...
fast_...
svnd...
line_b...
http.c (LOC: 1394, ...
daemon.c (LOC: 1369, Comp...
dir.c (LOC: 150...
pretty.c (LOC: 1...
sha1_name.c (LO...
graph.c (LOC:...
commit.c (LO...
convert.c (LOC: 132...
notes.c (LOC: 1298...
wt-status.c (LOC: 12...
combine-diff.c (LOC...
transport.c (...
cache.h (LOC: 1279, Com...
transport-helper.c (LOC: 1...
date.c (L...
sequenc...
submodule.c (LOC:...
bisect.c (LOC: 996, ...
fetch-pack.c (...
setup.c (LOC: 862, Comp...
log-tree.c (LOC: 856, Co...
tree-walk.c (LOC: ...
git-compat-util.h (...
http-walker...
git.c (LOC:...
streaming.c (LOC: 546, ...
remote-curl.c...
upload-pac...
attr.c (LOC: ...
http-backend.c (L...
parse-options.c (...
quote...
diff-lib.c (LOC: ...
utf8.c (LOC: 51...
notes-merg...
kwset.c (LO...
run-comma...
diff-delta...
strbuf.c (...
bundl...
wrapper.c (LO...
column.c (LO...
archive-zip.c (LOC:...
ll-merge.c ...
ident.c (L...
rerere.c (LOC: 7...
cache-tree.c (LOC: 70...
archive-tar.c (LOC: ...
diffcore-rename...
path.c (LO...
connect.c ...
archive.c...
sh-i18n-...
help.c ...
ws.c (...
pack-...
mat...
fsck.c (LO...
sym...
tree...
walker.c (...
list-o...
csum-file.c ...
pack-re...
builti...
pack...
name-has...
sideba...
dir.h (LOC: ...
credential...
xdiff-interf...
grep...
http...
trace...
refs.h (...
gpg-int...
block-sha1
usage...
ge...
diffcor...
remote-testsvn.c ...
sha1.c (L...
userdiff.c (...
tra...
pa...
pager.c ...
bas...
url...
test-c...
diff.h (LOC:...
remote...
repla...
sen...
notes.h (L...
parse-op...
strin...
obje...
resolv...
tag.c (LOC...
string-list...
reflog-walk...
diffcore-br...
merg...
hash...
strbuf...
exec...
cre...
crede...
prelo...
branch.c (LOC: ...
color.c...
entry.c (LOC: 2...
tes...
pat...
ht...
notes...
prom...
threa...
attr...
lockfile.c (...
sha1-loo...
ppc
diffc...
bul...
delta.h (...
not...
pac...
all...
get...
environm...
diff-no-...
credentia...
shallow...
test-d...
progres...
zlib.c (L...
parse...
pkt-li...
revisi...
test-li...
dec...
patch-...
mailma...
levens...
conver...
quote...
commit.h...
rea...
ser...
tree.c (L...
abs
diffc...
shell.c (L...
obj...
writ...

What can you visualize with this?



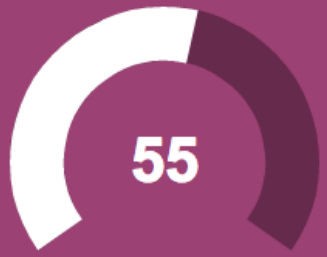
- Complexity
- Defects
- Static analysis results
- Churn
- Understanding
- Maintainability

Hello

This is your shiny new dashboard.

Protip: You can drag the widgets around!

Synergy



Last updated at 16:00

Buzzwords

Exit strategy	11
Leverage	3
Synergy	10
Turn-key	11
Web 2.0	4
Streamlininess	16
Pivoting	17
Enterprise	16
Paradigm shift	12

of times said around the office
Last updated at 16:00

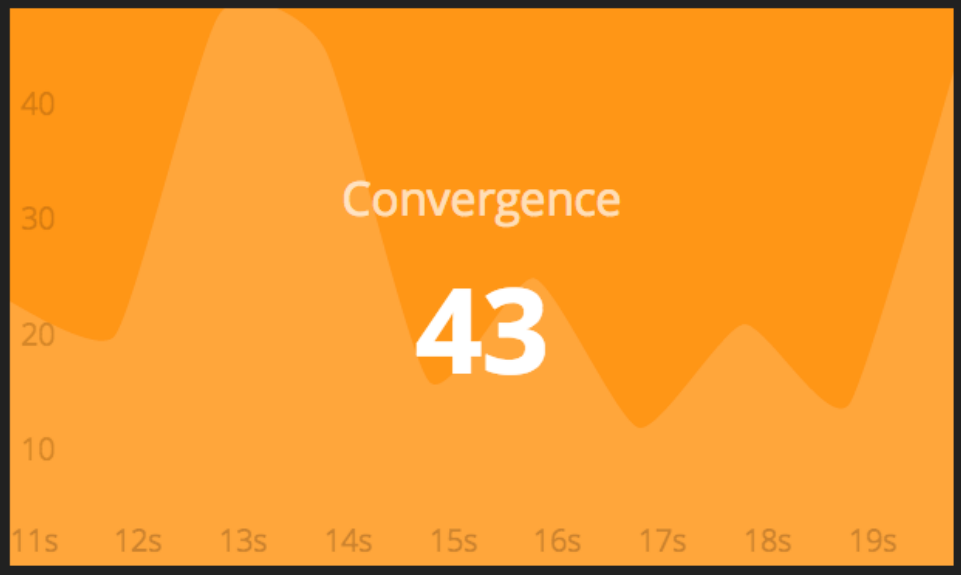
Current Valuation

$31

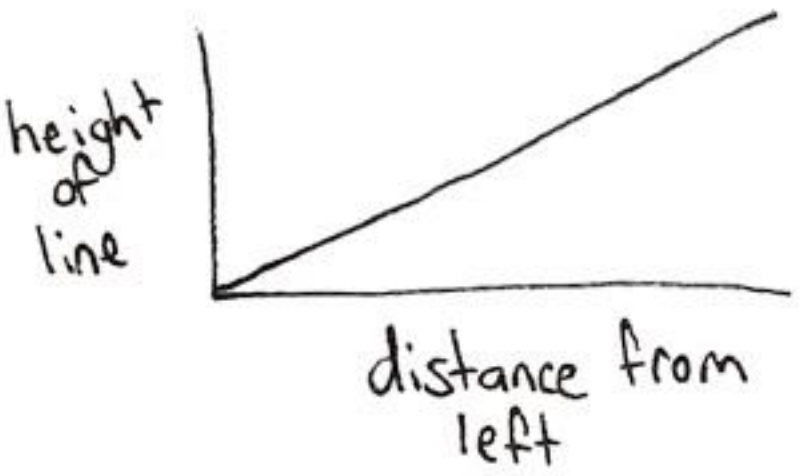
↑ 288%

In billions
Last updated at 16:00

Convergence



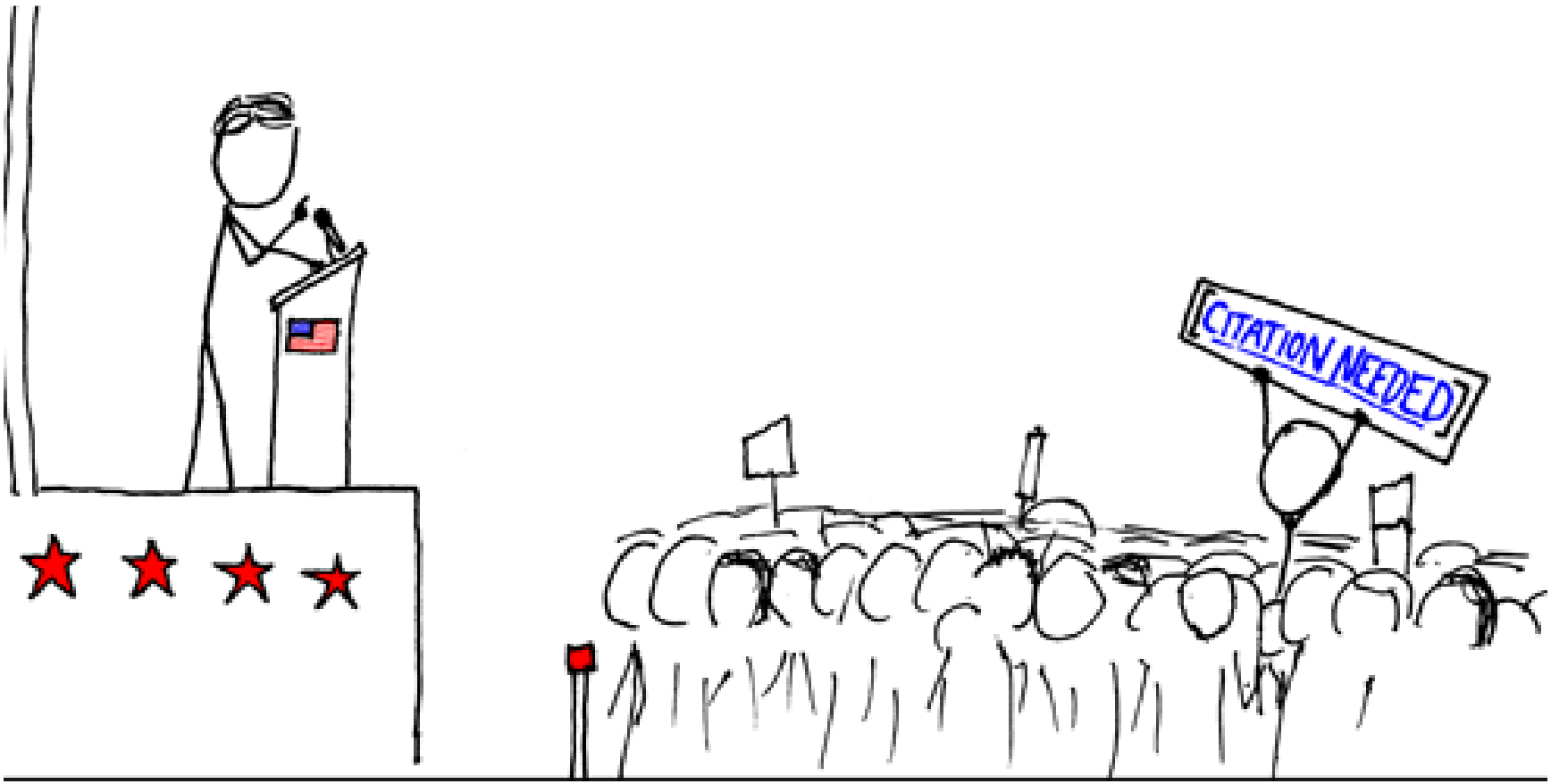
“Up and to the right”



- Trends are important
- Only trend metrics that will change (no-one is motivated by code coverage going from 2.066 to 2.067%)
- Put the goal or next milestone in the trend
- Treemaps can also show trends

Making the business case

“We need to fix this” [Citation Needed]



Quantify, visualize, communicate

- Quantify:
  - Communicate in terms of measured business costs and business benefits
- Visualize:
  - A picture is worth a thousand bug reports
- Communicate:
  - blah, blah, consultant, blah

Metrics

- Hard:
  - Complexity
  - coverage
  - test cases
  - churn, turmoil, changeset frequency
  - static analysis, dead code, duplicate code, warnings,
  - Defects (escaped)
- Soft:
  - Feature implementation time
  - Customer satisfaction
  - Employee satisfaction

Avoiding metricide

"If you can't measure it you cant manage it"

- The Managers Delight:

Hard conversations

- “We’re going slow now, but to go faster we need to slow down”
- “As we improve the codebase we will introduce regressions in seemingly random ways”
- “Throwing money at the problem is not going to significantly move the needle on quality”

How much effort to spend on Quality?

- Hint: you don’t get to decide

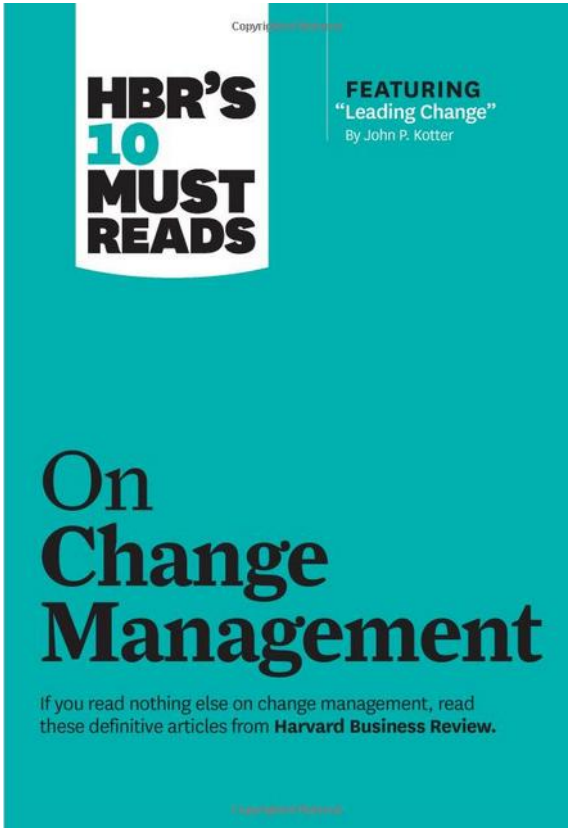
- You are going to have to sacrifice feature development
- in exchange you must promise more frequent release quality increments

Managing Legacy Restoration

Legacy Restoration is Culture Change

- Change can’t happen without new values
- New values must be driven by new culture
- Identifying waste requires new perspective
- From Complaining to Fixing

Phases to change



1. Establish a sense of urgency
2. Create a guiding coalition
3. Develop a vision for the change
4. Communicate the vision for buy-in
5. Empower broad-based action
6. Generate short-term wins
7. Never let up
8. Incorporate changes into the culture

Erik Schlyter

Quality by Being Careful

There seems to be a de-facto mentality throughout the industry within organizations that lack an established strategy to achieve and maintain quality. I call it "quality by being careful" and it is the simplistic comprehension that you can achieve quality and reduce defects simply by being careful enough. My concern is not that people use it as a principle, but rather that people use it as a base for the assumptions that guide their choice of process strategies. Although there is clear value in being mindful about your code, I will argue that this mentality in its pure form is incomplete, doesn't scale, and can be downright counterproductive in the context of legacy systems or projects where inferior test coverage and lacking requirements is the norm.

Tuesday, 25 September 2012 - 15:10 | Permalink | 2 Comments | Read Later

It comes natural to be careful when you got something at stake, and generally I would not suggest there is something wrong with it. It makes you focused and it might make you avoid sloppiness, but I consider it important to acknowledge the limitations of this mentality when you are collaborating with a team on a shared codebase. In the context of software development I first encountered this way of thinking as early as the first assignments at the

Thoughts, ideas and articles

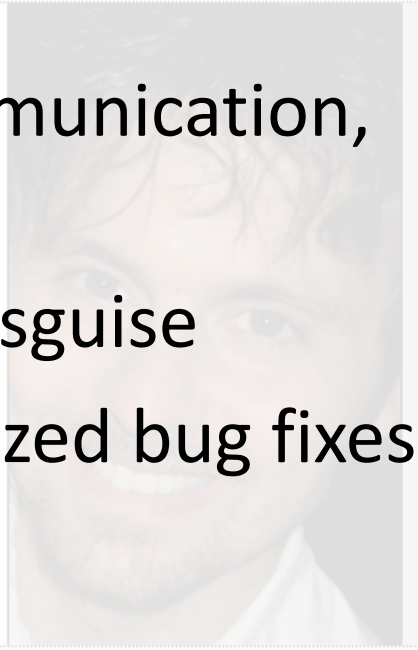
2012-09-25
Quality by Being Careful

2012-08-14
The Evaluation of My Software Sabbatical

2012-06-11
Software Sabbatical

Quality by Being Careful

- Defects come from incomplete communication, not only mistakes
- The need for carefulness is fear in disguise
- Fear of changing code leads to localized bug fixes



✓ Write clean and understandable code

✓ Peer review

✓ Different levels of specification and testing

Improve both trust and quality

- Without trust you can’t
  - get the sponsorship to invest in internal quality
  - have the developers believe in a better future

- Trust comes with transparency

Lottery Factor

Not my stuff

- Large long-lived codebases rarely have many of the original developers around
- This makes the current developers code archeologists
- Ownership comes before collective ownership
- Engineers are historians

Engineers as historians

- The larger a codebase is, the more of it will be completely unknown to the development team, especially if the project went through a period of explosive growth.
- Developers are more likely to re-implement functionality that already exists
- There will be a lot of cargo cult programming. Existing patterns in the codebase will be copied without knowing the true reasons behind doing so. This is dangerous because even the bad practices and leaky abstractions will be followed and cemented

Knowledge and Skills

- Knowledge sharing
- Lottery factor
- Skills matrix
- Code understanding
- Pair programming
- Code reviews
- Learning culture

Where to start: churn + roadmap

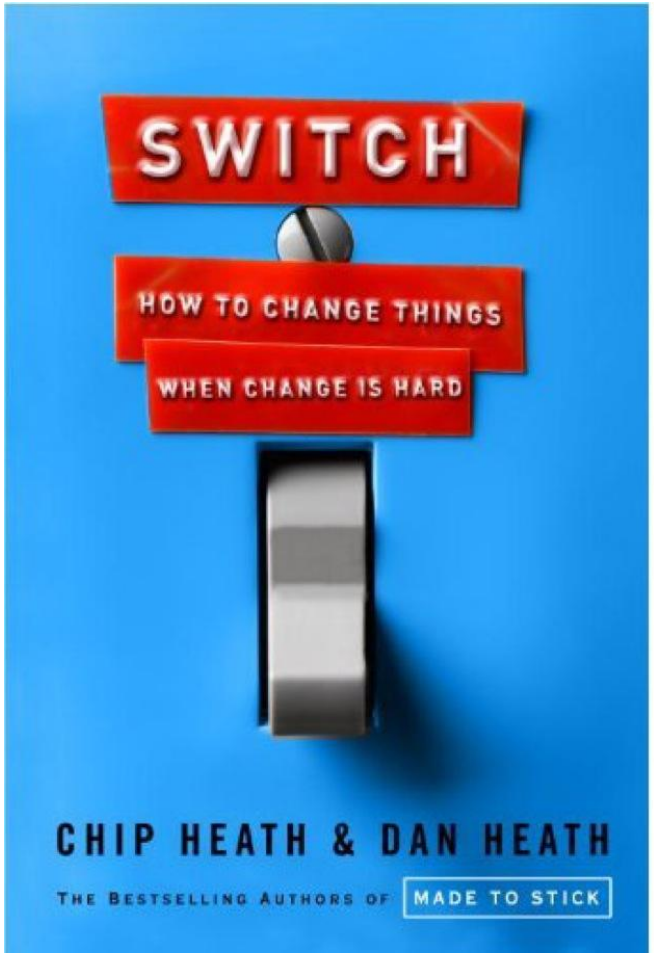
- Combine what is changing with your new development roadmap
  - Things that don't change:
    - don't have bugs
    - are not slowing down feature development
    - are probably rarely read
  - Compare new feature only development with refactoring + new development
    - improving areas of high churn will h

Is it *really* worth it?

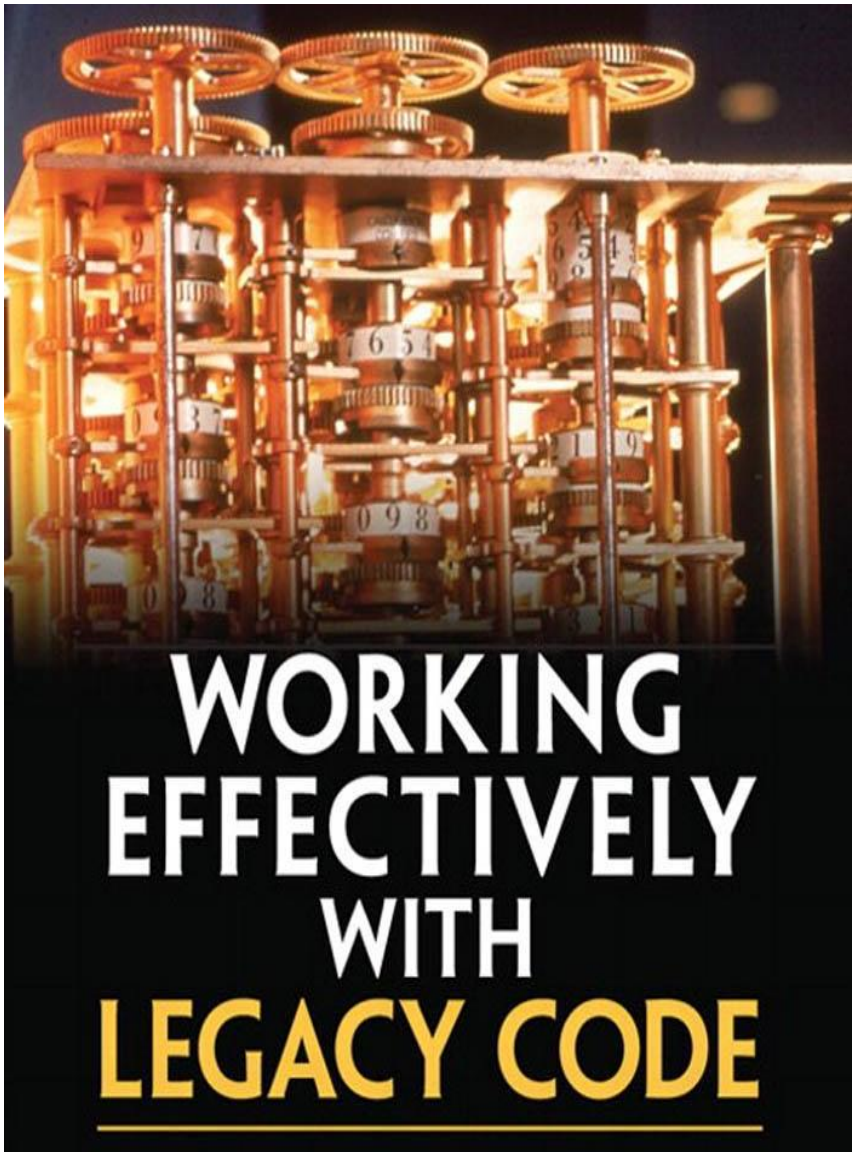
- In making the case for remedial work, you need to justify the cost – it might really be cheaper for the company to have crappy code.
- If this is the case, you might want to look elsewhere for work. There are no happy endings here
- The good news is that this situation is highly unlikely

Resources

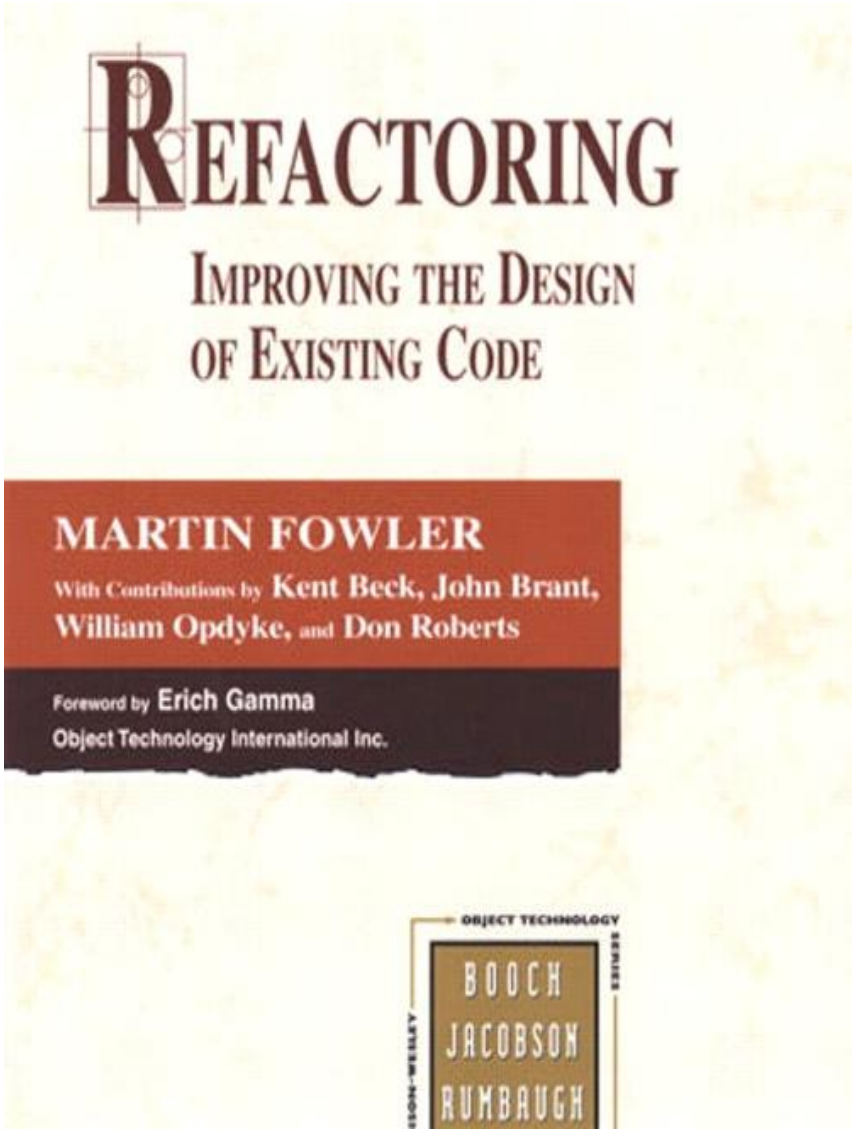
Change should be managed



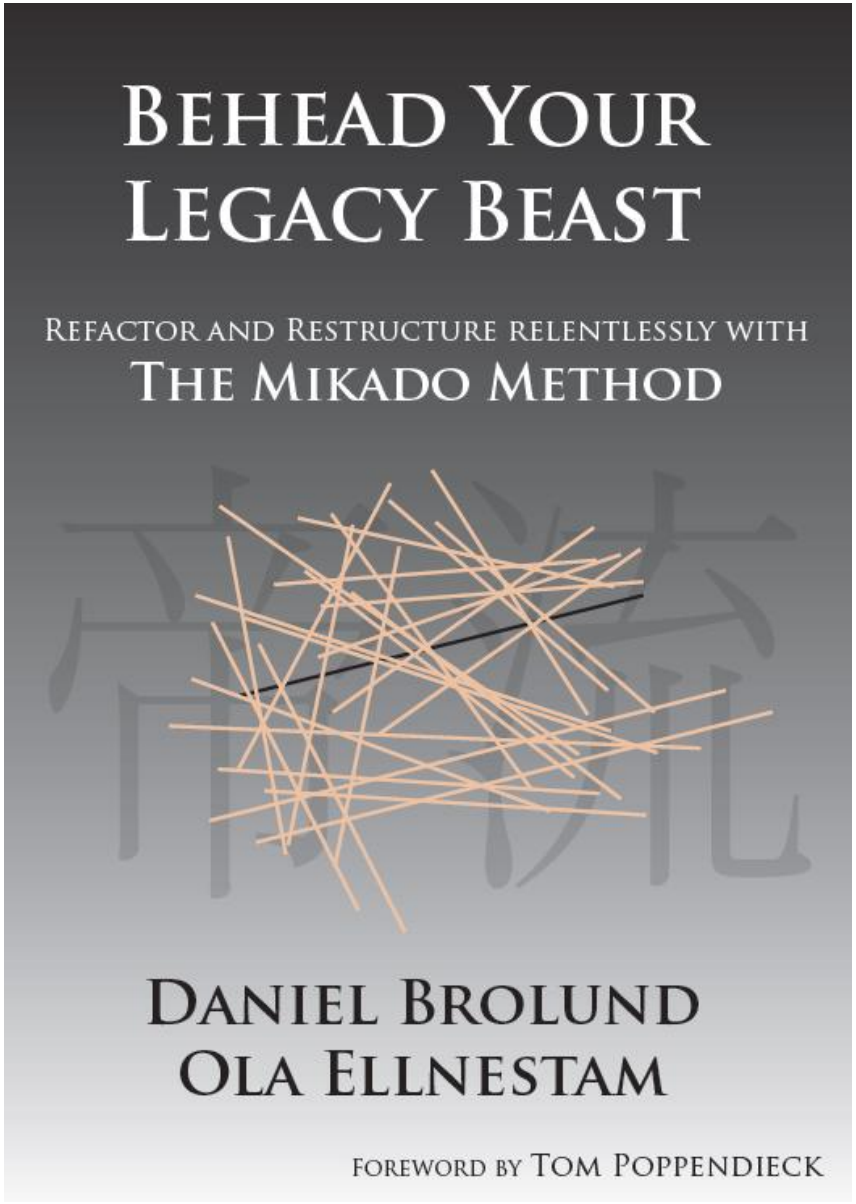
- Direction
  - Look for the bright spots, think in terms of specific behaviors, be specific in goals
- Motivation
  - Make people feel the need for change, make it small, cultivate a growth mindset
- Environment
  - Change the situation, build habits, and rally the herd



- How to get legacy code under automated test
- How to break dependencies
- Strategies for dealing with common anti-patterns



- Detailed explanations of 70 refactorings together with the mechanics of how to apply them safely



- A **structured** technique for avoiding the weeds when performing deep refactoring

LibreOffice: the story of cleaning and re-factoring a giant code-base

Michael Meeks <michael.meeks@suse.com>
mmeeks, #libreoffice-dev, irc.freenode.net

"Stand at the crossroads and look; ask for the ancient paths, ask where the good way is, and walk in it, and you will find rest for your souls..." - Jeremiah 6:16

openSUSE

LibreOffice

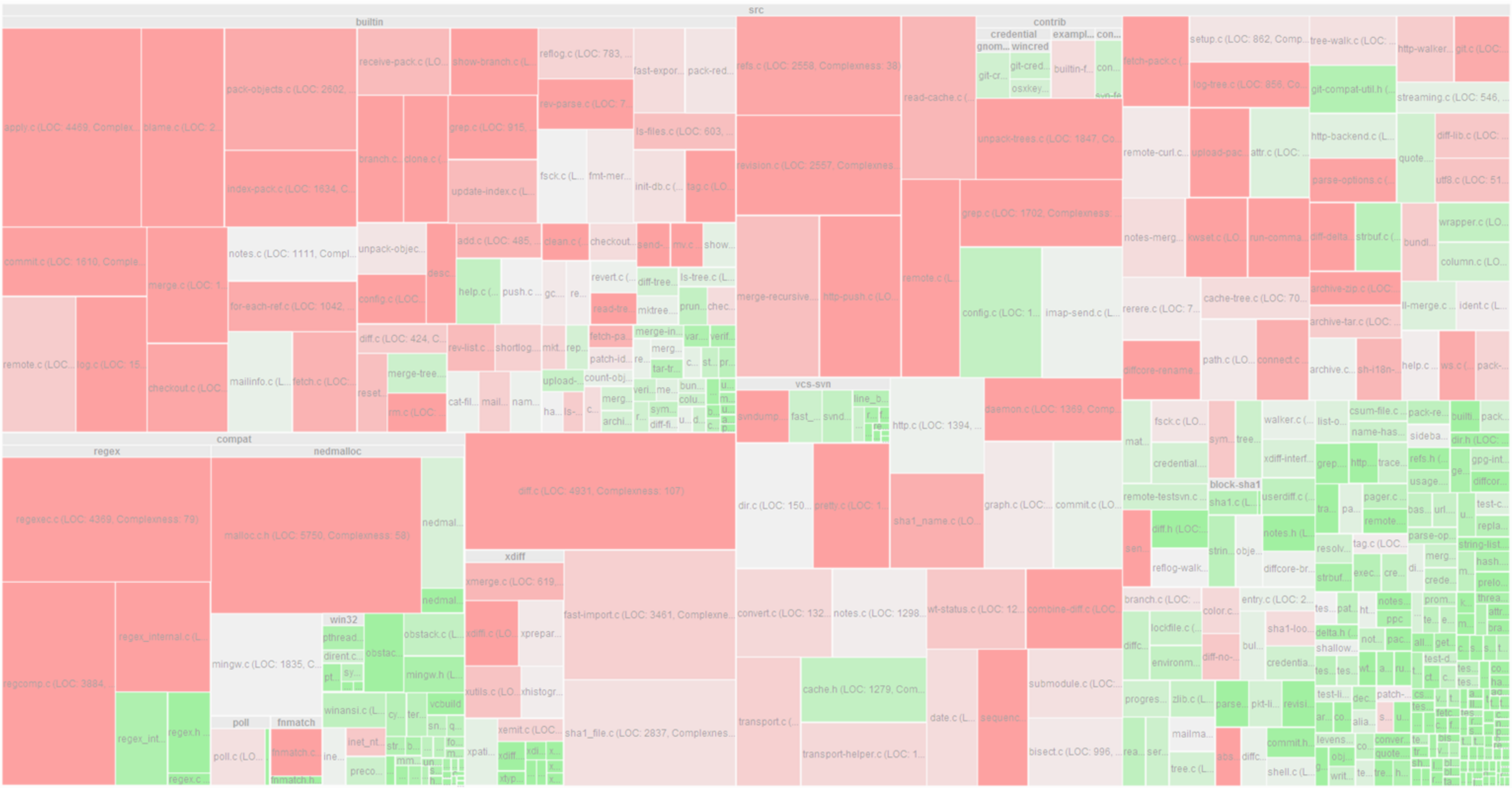
Conclusions

Conclusions

- Prevention is better than cure
- Legacy software is valuable software
- There is always a business case for restoration
  - You just need to prove it
- Restoration is culture change

Too big to fail?

Questions?



src
builtin
contrib
compat
regex
nedmalloc
xdiff
vcs-svn
win32
poll
fnmatch
credential
block-sha1
apply.c (LOC: 4469, Complex...
pack-objects.c (LOC: 2602, ...
index-pack.c (LOC: 1634, C...
commit.c (LOC: 1610, Comple...
notes.c (LOC: 1111, Compl...
for-each-ref.c (LOC: 1042, ...
refs.c (LOC: 2558, Complexness: 38)
revision.c (LOC: 2557, Complexnes...
unpack-trees.c (LOC: 1847, Co...
grep.c (LOC: 1702, Complexness: ...
daemon.c (LOC: 1369, Comp...
http.c (LOC: 1394, ...
regexec.c (LOC: 4369, Complexness: 79)
malloc.c.h (LOC: 5750, Complexness: 58)
diff.c (LOC: 4931, Complexness: 107)
fast-import.c (LOC: 3461, Complexne...
sha1_file.c (LOC: 2837, Complexnes...
mingw.c (LOC: 1835, C...
cache.h (LOC: 1279, Com...
transport-helper.c (LOC: 1...
notes.c (LOC: 1298...
bisect.c (LOC: 996, ...
setup.c (LOC: 862, Comp...
log-tree.c (LOC: 856, Co...