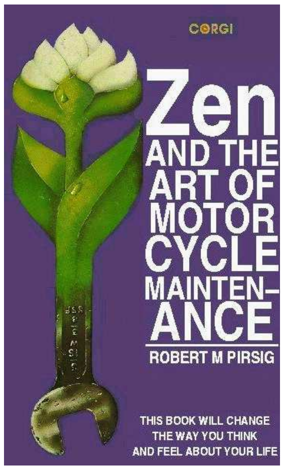
Zen and the Art of Software Maintenance

CBC.ca WebEd
David Tilbrook
dt@qef.com
Dec. 1st & 2nd, 2003

Zen & the Art of SW Maint.
Preface


CORGI
Zen AND THE ART OF MOTOR CYCLE MAINTEN-ANCE
ROBERT M PIRSIG
THIS BOOK WILL CHANGE THE WAY YOU THINK AND FEEL ABOUT YOUR LIFE

"Working on a motorcycle, working well, caring, is to become part of a process, to achieve an inner peace of mind."

Robert M. Pirsig

Zen & the Art of SW Maint.
Overview

- Objectives of this Presentation
- The Pursuit of Quality
- Project Failures
- Software Hygiene
- Rolling Release Engineering
- Large Scale Project Maintenance and the Individual Programmer's Role
- Architecture for a Well Maintained System
  - The Principle of Locality
  - Source Organization and Work Flow
  - The Gatekeeper
- The CORE (Controlled Requirements Evaluation) System
  - Introduction
  - Break Out session exercise
- The Development Process
  - Change Control
  - Testing

Zen & the Art of SW Maint.
Schedule

Day 1

● 1:00–2:30	Introduction, Software Hygiene, Architecture, ...
● 2:30–3:00	Break
● 3:00–3:30	CORE and Break Out Instructions
● 3:30–4:30	Break Out Exercise
● 4:30–5:00	The Development Process

Day 2

● 1:00–2:00	Team Presentations & Concluding Remarks

Zen & the Art of SW Maint.
Objectives of this presentation

- Understanding causes of failure
- Appreciation of hygienic approaches
- Examining the process
- Exploring your role
  - Encourage you to think about how you work and play with others
- Attributes of a good process

Zen & the Art of SW Maint.
The Pursuit of Quality

- Quality is elusive
- Q.A. necessary but not sufficient
- Need to ensure that individual efforts properly incorporated.
- Need to ensure that developers' intentions are realized in delivered product
  - assuming of course they understood the requirements and specification
- Need to ensure product installed flawlessly
- Need to ensure testing is valid
  - What is being tested is what will be delivered.
  - What is delivered has been tested
  - Testing a system that is not delivered is a waste of resources
- Q.A. should be proactive rather than reactive.

Zen & the Art of SW Maint.
Project Failures

- Quality jeopardized by complexity of process involving large team
  - Management is complex
  - Assuring that work being done is relevant is difficult
  - Many problems introduced during integration
  - Problems arise due to discrepancies in environments
  - IBM study showed that > 65% of bugs introduced during bug fixing
- Version skew a constant threat
- Distributed development necessary but complicates ensuring appropriate environment
- Deficient construction approaches can invalidate developer's testing.
  - What they are testing is not what will be delivered.

Zen & the Art of SW Maint.
Software Hygiene

Purpose

- To maintain healthy projects
  - through application of simple principles and practices.

Difficulties in projects

- Often not the result of deep technical problems
- Often due to lack of basic hygiene
  - failure to co-ordinate activities and products
  - principles of hygiene violated

Zen & the Art of SW Maint.
Why Bother?

Caveat

- Hygiene itself will not ensure success
- Must have effective tools and methods
- Hygiene must be at the heart of the software process

Ultimately

- Nothing spectacular from its presence
- Effects of absence can be failure

Zen & the Art of SW Maint.
SW Hygiene Principle (1)

Principle 1

Everybody involved in the project should know the objectives of what he or she is doing.

- know what you are trying to accomplish
- if requirements are ill–defined, problems will arise if people act as if they are well–defined

Zen & the Art of SW Maint.
SW Hygiene Principle (2)

Principle 2

Achievement of the overall project objectives should follow immediately from achievement of all individual objectives.

- seamless integration
- rigourous testing
- good communication

Zen & the Art of SW Maint.
SW Hygiene Principle (3)

Principle 3

Both individual objectives and overall project objectives should be realistic.

- no factor more damaging than unrealistic goals
- people are invariably overly optimistic about what they can do and especially how quickly they can do it
  - many do not realize that an eight–hour work–day does not mean eight hours of productive work

Zen & the Art of SW Maint.
SW Hygiene Principle (4)

Principle 4

There should be a known method for addressing each individual objective.

- quick and dirty leads to costly and buggy in the long term
- do it right the first time and if you don't know how to do it right:
  - steal from someone who does, or
  - take time to design solution
  - doing it right usually should mean doing it once and only once

Zen & the Art of SW Maint.
SW Hygiene Principle (5)

Principle 5

Changes should be controlled, visible, and of known scope.

- make sure people understand use and purpose of the versioning system
- need to be able to pre–determine what impact a change will have on entire product

> 50% of cost of software is due to change

> 65% of bugs introduced during changes

Zen & the Art of SW Maint.
SW Hygiene Principle (6)

Principle 6

Both people and products should be insulated from the effects of changes that are not (currently) of relevance.

- Others should not see a change until it really is ready and has been tested.
  - Testing should be comprehensive enough that all those affected (within reason) are covered.
- Try to reduce visibility of details not relevant to others
  - e.g., in C everything that can be static should be static
- Ensure that change that is relevant is done so that those affected will react or be informed properly
  - use centrally declared prototypes
  - use most pedantic options and eliminate all warnings

Zen & the Art of SW Maint.
SW Hygiene Suggestion (1)

Suggestion 1

Focus on the process as a whole, rather than on the (final) product.

- hygiene is an attribute of the process by which the product is produced

Zen & the Art of SW Maint.
SW Hygiene Suggestion (2)

Suggestion 2

Invest more effort in "higher level" descriptions.

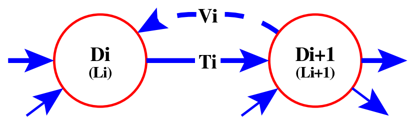
- test your descriptions
- design mechanisms whereby your implementation of a description can be verified
- too many people still rush into code
- the earlier a bug is specified, the more costly it is
- misinterpretation of descriptions can jeopardize an entire project

Zen & the Art of SW Maint.
Descriptions?

A common mistake is to use terminology such as "requirements analysis", "design", "specification", "implementation", etc.

- Such terminology gets in the way.

The Software Transactional Model



Description Di in language Li transformed using transform Ti into description Di+1 in language Li+1 and then verified using Vi

- A transform T could be stepwise refinement, recursive descent, decomposition, composition, a tool (e.g., cc), a human interpretation
- A verification V could be a human review, a testing process, ...

Zen & the Art of SW Maint.
SW Hygiene Suggestion (3)

Suggestion 3

Co–ordinate activities as well as products.

- Too much focus on CM
- Not enough focus on activities
- Need to evolve common model encompassing both.

Zen & the Art of SW Maint.
SW Hygiene Suggestion (4)

Suggestion 4

Adopt a strict policy on project phases.

- Expectations are usually unrealistic
  - see suggestion 2
- Not always possible to judge realism of objectives at project start
- Can compensate by sub–dividing project into self–contained phases
  - sub–projects should themselves be treated as separate projects
- Scheduling and timelines adjusted as sub–phases near completion
- Client may be reluctant to accept open–ended scheduling, but it is in the client's own best interest.

Zen & the Art of SW Maint.
SW Hygiene Suggestion (5)

Suggestion 5

Provide more semantic information to the configuration management and build systems.

- Need to extend prerequisite relationships beyond the syntactic if possible.
- Software CM needs to be more consistent with modern approaches in other areas

Zen & the Art of SW Maint.
Conclusions

Poor hygiene is prevalent since people are usually too engrossed with technical demands and details.

They lose sight of big picture.

Ask yourself:

- are objectives for activities clearly and precisely defined?
- are there activities for which objectives are unrealistic?
- how are results from individual activities combined to meet objectives of overall project?
- are there effective mechanisms for coordinating changes and limiting scope of their impact?

Zen & the Art of SW Maint.
Software Nostrums

- Give a damn
- One and only one source
- The source is the product
- Ensure production complete and deterministic
- Ensure everyone understands process
- Test, test, and test again
- Avoid simultaneous dramatic changes
- Adopt standard project architecture
- Do post mortems on bugs and failures
- Remember two primary objectives:
  - have fun
  - make money (or whatever the equivalent is at the people's network).

Zen & the Art of SW Maint.
Rolling Release Engineering

A major objective of process is to provide the mechanisms whereby a product is maintained in constant readiness to be released on demand.

An important attribute of release readiness is the assurance that the product, as built, released, and shipped today, can be reproduced exactly at some future date by simply extracting the appropriate source files from the version system archives and rebuilding the product in its entirety.

The use of incremental builds can be an impediment, but the following can help:

- the separation of the baseline and working source trees
- bare metal builds
- file system integrity checks

Zen & the Art of SW Maint.
System Architecture: Intro

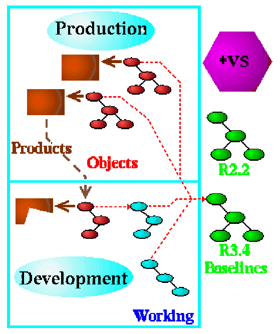
A Well Maintained System's Architecture

- Desirable Characteristics
  - Clean production
  - Consistent w.r.t. product sources, installation, and testing
  - Adaptable to changing requirements
  - Universally helpful w.r.t. support and maintenance
- Goals:
  - Extendable
  - Reusable
  - Compatible
  - Portable
  - Verifiable

Zen & the Art of SW Maint.
Large Project Organization

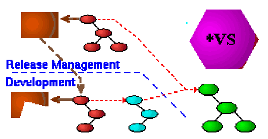
- Single rooted tree
  - Root contains directories for admin, testing, and component directories
- Well partitioned into components that can be built independently
  - See Principle of Locality (see slide 30)
- Single pass build, except may have separate passes for Man and Post processing
- Dependencies on other projects to be satisfied by those projects' installed products – do not reference their source.

Zen & the Art of SW Maint.
Work Flow



*VS	The version system vault
R3.4	A baseline extracted from *VS
Working	A developer's working tree, preferably containing changed files only
Objects	The build trees
Products	The installed products

Zen & the Art of SW Maint.
Work Flow (2)



- Release management maintains a baseline and installed release based on it.
  - Should be separate products for each required and active configuration.
- Each developer extracts files to be changed into their own sparse working tree (a.k.a. sandbox). New files created in working tree.
  - Development builds are done in separate object tree, referring to baseline for missing sources, and installed product for missing prerequisites.
- Developer submits changes to *VS through approval or Q.A. process after which changes will appear in the baseline, thus becoming visible to other developers and Release Management.

Zen & the Art of SW Maint.
The Gatekeeper

Approval Process??

- See Principles 5 & 6

Need to make changes visible in controlled and non–interfering manner as requirements dictate.

- If your software is mission critical, changes should be reviewed and tested thoroughly before going Prime Time.
- If your changes fail, your primary responsibility is to fix them A.S.A.P.
  - Correct immediately, or
  - Back 'em out!
- Gatekeeping role should be responsive enough to encourage and support small changes to minimize time required for submission and approval.
  - Easier to test.
  - Easier to find and fix problems introduced.
- Approval process should not delay further development.

Zen & the Art of SW Maint.
The Principle of Locality

Within any sub–project:

- Prerequisites are limited to installed product or self references.
  - May not refer to another project's source or uninstalled byproducts
  - Must use installed version as would exist for installation up to that sub–product.
- Dependencies limited to sibling directories
  - i.e., ../auntie/...
  - Try to avoid references to top level directory's contents of its sub–directories' contents.

Adherence to this principle goes a long way in supporting and encouraging:

- highly partitioned development
- incremental construction
- simple single pass construction
- reduced (but still legitimate) testing
- elimination of cyclic dependencies
- efficient builds

Zen & the Art of SW Maint.
F.S.I.C.

Nothing can jeopardize quality or delay progress more than a missing or lost file!

Oh, except for a file that shouldn't be there!!

- The major problem faced by the Y2K projects was the inability to rebuild products due to the loss of the source.
- The existence of a file that shouldn't be there will:
  - obscure the loss of source
  - obscure cyclic dependencies
  - obscure build deficiencies
  - possibly change the system's semantics

File System Integrity Check

- Need process and supporting databases to check integrity of source, object, and product trees.
- Source tree check should be run frequently.
- Object and product trees should be checked as part of the build.

Zen & the Art of SW Maint.
F.S.I.C. (2)

The Source Database

- Manifest of administered files
- List of old sources – in VS but obsolete
- List of unadministered sources (as small a list as possible)
- List of known temporary files
- Conventions (patterns) for temporaries e.g., "core.*", ",*", ...

Source Check

- Produce lists of:
  - potentially new sources – exist but not in manifest and not known or recognized temp.
  - missing files – in manifest but does not exist.
  - files being edited
  - recognized temporaries
- In case of error, take corrective action (update database, create VS admin files, remove files, ...) and repeat.

Zen & the Art of SW Maint.
F.S.I.C. (3)

The Object/Product Database

- Superset list of object and product trees
- Exceptions for configurations
- Normal deviations (e.g., man/cat?/*.1)
- Conventions (patterns) for temporaries e.g., "core.*", ",*", ...

Object/Product Check

- Produce lists of:
  - potentially new files
  - missing files – in database but does not exist
  - recognized temporaries
- In case of error, take corrective action (update database, remove files, ...) and repeat!
- Use bare–metal and parallel builds to check databases for discrepancies

Zen & the Art of SW Maint.
End of Part 1

Break

- Please be back by 3:00

Zen & the Art of SW Maint.
The CORE System

Controlled Requirements Evaluation

CORE is a viewpoint perspective analysis which can be applied on most types of systems e.g., human, computer, mechanical

- system is partitioned into components, roles, or viewpoints
- viewpoints are analysed w.r.t. to their consumer/producer relationships with other viewpoints, determining their:
  - from (F) – sources of inputs
  - input (I) – the required inputs
  - processing (P)
  - outputs (O) – processing products
  - to (T) – destination or consumer of outputs

  referred to as F.I.P.O.T. table.
- Analysis attempts to ensure connectivity of collected tables.
  - if Alpha produces X to be sent to Beta, then Beta should be expecting to receive X from Alpha.

Zen & the Art of SW Maint.
CORE: Introduction

Actually that's it.

All you need now is to learn how to create the first cut at the tables and then how to evolve and debug the resulting directed graph.

- But we'll do that as an exercise.

Break Out Exercise

You and your group represent one viewpoint.

In the hour allotted your group will:

- pick a chair to keep your group to schedule
  - an hour isn't very long! If pick takes more than 2 minutes, flip a coin.
- pick a presenter for tomorrow's 8–10 minute presentation of your group's results
- formulate a brief mission statement for your group's viewpoint

Zen & the Art of SW Maint.
CORE: Break out session (2)

- Create the FIPOT table for your view on flip charts.
  - Think about what you consume/produce, would like to have or do, or should do.
  - Don't worry about getting it right – you can't without seeing the other viewpoints.
  - Don't bother editing at this time.
  - Don't try to connect column entries – just list them.
  - Try to write legibly so that table can be read from anywhere in lecture room.
- Ensure that digital picture of your results taken and mailed to Derek and dt@qef.com before removing the flip charts.
- Presenter (with help if necessary) should prepare single OHP slide of mission statement and the FIPOT table for presentation tomorrow afternoon.
  - You will post your flip charts on the lecture room wall.

Zen & the Art of SW Maint.
CORE: Break out session (3)

Process to be considered is a production release of an new system to handle NHL playoffs.

- Limit consideration to later phases
  - assume that early project descriptions are in place
  - Implementation is well underway and full project integration almost ready.

Viewpoints are:

- Production & Operations Management
- Release Management
- Project Team Leaders
- Quality Assurance
- Developers

Zen & the Art of SW Maint.
The Development Process

- Development is a lot more than cutting code.
- The code cut must be:
  - documented,
  - tested, possibly creating new tests,
  - delivered with documentation describing changes,

  in such a way that if doesn't interfere with other teams' work.
- Developers must view their product as being the source, that is delivered in a timely and expedient manner to the build team for smooth and effective integration into the code baseline.
- When source is delivered, the submitting developer is making a statement that the code is ready for publication and making commitment to support (i.e., fix) his/her changes.
- Saying that it "works for me" is irrelevant.
  - It has to work for your client (the build team).

Zen & the Art of SW Maint.
Development: Change Control

CBC.ca source is under CVS control

- Why? What does it do for you?
- Do you know how to perform the necessary functions w.r.t. source?
- And are you sure that everybody else uses the same functions?
- Are these practices safe? foolproof?
- Do you have a mechanism whereby developers can use a *VS to save progress without interfering with others?
  - First two points of SDG 4.2 might be contradictory or counter–productive.
- And why is something as important as source management left until section 4?
  - The use or spaces in code is in 1.1!

Jeremy, good stuff. Far better than I have seen at many other organizations.

My quibble is that coding styles are an appendix to the "How We Get Things Done" guide.

Zen & the Art of SW Maint.
Development: Testing

More Gratuitous Quibbles with the SDG

- Must ensure objectives for and overall approach to Q.A. clearly understood
- Must emphasize that testing must be designed and encoded into system from outset.
  - Q.A. group must be involved from very beginning.
- Quality of testing itself must be validated.
- Make testing as easy as possible to do.
- Make sure testing is audited.
- Ask for help doing your testing.
  - rob@iron (mips) found a problem that dt@gold (mips), dt@silver (sun), dt@chlorine (sco – a noxious gas) didn't find, but dt@argon (mips) did.

Zen & the Art of SW Maint.
Conclusions & Q&A

"We want to make good time, but for us now this is measured with the emphasis on "good" rather than on "time" and when you make that shift in emphasis the whole approach changes."

Robert M. Pirsig
pp4, *Zen and the Art of Motorcycle Maintenance*

Pirsig was writing about taking secondary roads, but the view is applicable to all sorts of human activities including software development.

- In fact in the realm of software an emphasis on "good" often leads to better (both qualitatively and quantitatively) times.