A Simple Matter of Configuration

Roger Orr

OR/2 Limited

How can we tame the complex world of configuration?

– ACCU 2010 –

A Simple Matter of Configuration

Death by Powerpoint

Roger Orr

OR/2 Limited

How can we tame the complex world of configuration?

– ACCU 2010 –

A Simple Matter of Configuration

*Death **to** Powerpoint*

Roger Orr

OR/2 Limited

How can we tame the complex world of configuration?

– ACCU 2010 –

A Simple Matter of Configuration?

- You've probably heard of SMOP (A simple matter of programming)
- “It's easy to enhance a FORTRAN compiler to compile COBOL as well; it's just a SMOP.”

A Simple Matter of Configuration?

- You've probably heard of SMOP (A simple matter of programming)
- “It's easy to enhance a FORTRAN compiler to compile COBOL as well; it's just a SMOP.”
- We have the same problem here.
- “It's easy to change your program to use Oracle rather than MSSQL, it's just a SMOC.”

A Simple Matter of Configuration ?

This is a SMOC(K)

What is the reality?

- Configuration is often complex

What is the reality?

- Configuration is often complex
- Sometimes over-complex!

What is the reality?

- Configuration is often complex
- Sometimes over-complex!
- There is no one-size-fits-all solution

- This PC has 730 ini/cfg/config files on it!

What is the reality?

- Configuration is often complex
- Sometimes over-complex!
- There is no one-size-fits-all solution
- As Albert Einstein famously said:

  “Make everything as simple as possible, but not simpler.”

The Complex Reality of Configuration

This is a CROC(K)

What *is* configuration?

- I found it hard to define the word: is it
  - Data?
  - Meta data?
- It seemed clearer to focus on intent:
  - “Setting up for a particular purpose”
  - "Configuration is then seen as a structured process which transforms the generic package into a system individualised for the organisation-specific context." http://is.tm.tue.nl/staff/wvdaalst/publications/p356.pdf

What *is* configuration?

- I found it hard to define the word: is it
  - Data?
  - Meta data?
- It seemed clearer to focus on intent:
  - “Setting up for a particular purpose”
- This is the root of the complexity – the **purpose** and the **setup** can vary widely

What is *not* configuration?

- Searching for “patterns for configuration of software” finds solutions to a different problem
- Sadly “software configuration management” has very little to do with the management of the configuration of software
- SCM focuses on the process of reliably producing software artifacts that meet their requirements: version control, change management, etc.

An example

- Let's take a simple program and see how it might be configured
- Doing this will help us identify some of the forces involved in configuration

Hello World

```
#include <iostream>


int main()
{
   std::cout << "Hello world" << std::endl;
}
```

Hello World #2

- Setup at compile time

```
#include <iostream>


int main()
{
   std::cout << "Hello Roger" << std::endl;
}
```

- “Magic numbers & literals are a configuration aspect of the code chosen to be implemented at compile-time.”- Jason McGuiness

Hello World #2

- Setup at compile time

```
#include <iostream>
#define _STR(X) #X
#define STR(X) _STR(X)
int main()
{
   std::cout << "Hello STR(NAME)" << std::endl;
}
```

```
cl /DNAME=Bill hello.cpp
```

Hello World #3

- Auto-setup from the environment

```
#include <iostream>
#include <cstdlib>
int main()
{
   char const * who = std::getenv("USERNAME");
   std::cout << "Hello " << who << std::endl;
}
```

Hello World #4

- Setup from the command line

```
#include <iostream>

int main(int argc, char **argv)

{

   char const * who = argc > 1 ? argv[1] : "world";

   std::cout << "Hello " << who << std::endl;

}
```

Hello World #5

- Setup from the user(s)

```
#include <iostream>
#include <string>
int main()
{
  while (std::cin)
   {
      std::cout << “Who are you?”;
      std::string who;
      std::cin >> who;
     std::cout << "Hello " << who << std::endl;
   }
}
```

Other directions

- These examples only focused on the **name**.
- Depending on the **purpose** other things might need to be setup
  - Language
  - Output destination
  - Presentation (font, size, colour)

Other directions

- These examples only focused on the **name**.
- Depending on the **purpose** other things might need to be setup
  - Language
  - Output destination
  - Presentation (font, size, colour)
- And that's just for “hello world”!

What have we learned?

- Configuration can be applied at many stages of the program, from during coding to at run time.
- Values can come from multiple sources
- Values may change
- Many technical solutions are possible

What needs configuring?

- A key step in deciding **how** to configure is to identify **what value types** need configuring.
  - What may change/what won't?
  - Who (or what) knows the required values?
  - When are they known?
  - Are they static or dynamic?
  - Mandatory or optional?

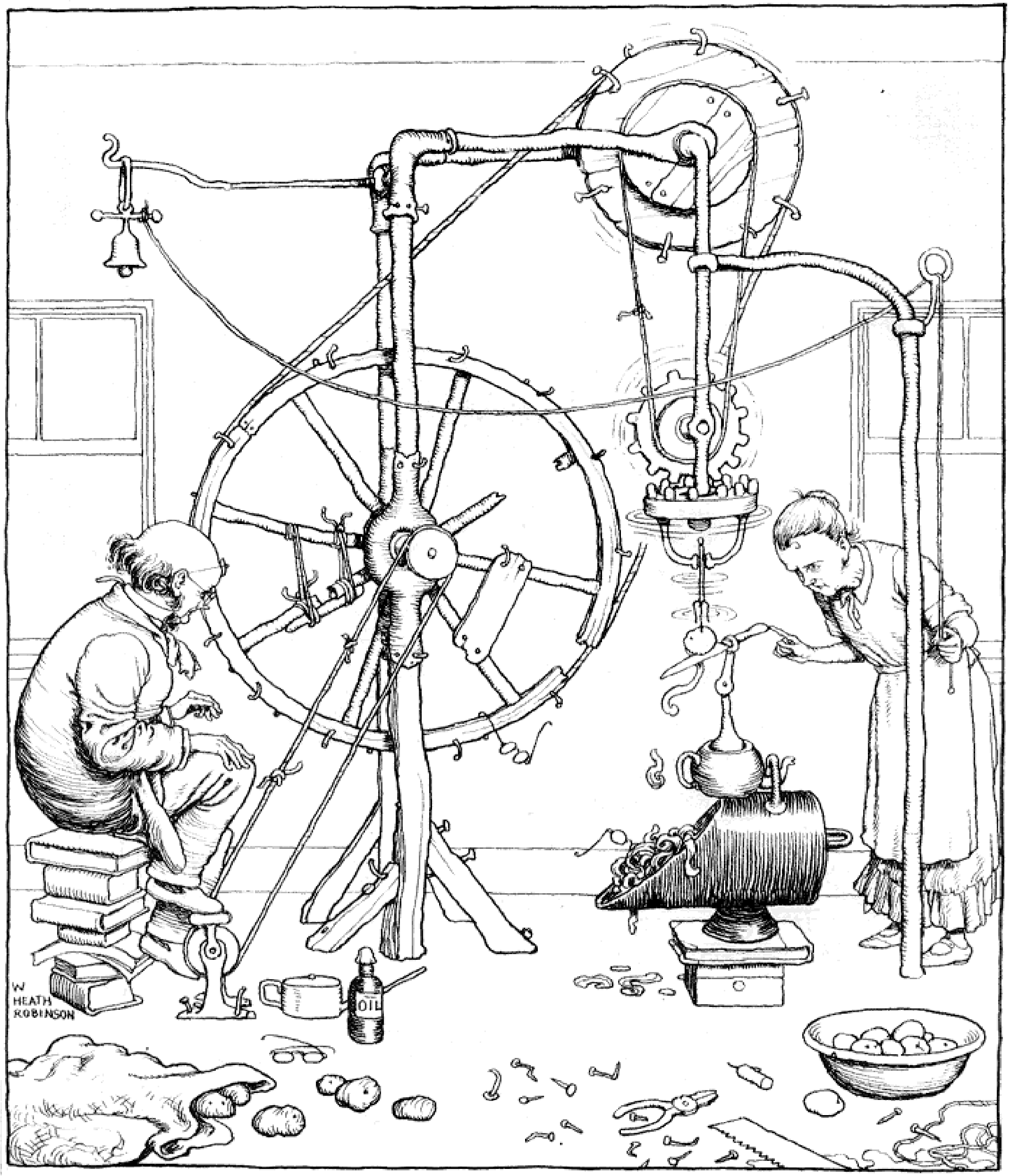
What may change?

- Processing configuration is costly
  - Code to read it
  - People to maintain it
  - Time to fix bad configuration
- Decide what should be configurable and what decisions you can/will make up-front

What may change?

- You've heard of
  - YAGNI (“You Ain't Gonna Need It”)
  - TAGRI (“They Ain't Gonna Read It”)

  I think configuration needs
  - NIGMI (“Nobody Is Gonna Modify It”)
- Don't need to make *everything* configurable



The Professor's invention for peeling potatoes.

Who knows the required values?

- Configuration values can come from many places, including:
  - “Extrinsic” data (e.g. support URL)
  - Installation data (e.g. OS version)
  - Runtime data (e.g. username)
  - Other systems (e.g. database)
  - The user

When are they known?

- During development
  - Can choose to code in or soft configure
- At installation
  - For example the "`./configure`" command
- At program start
  - The data may be provided by many sources, including command line arguments, property files and database queries

Are they static or dynamic?

- Can configuration values change during the execution of the program?
- If so, the program is more flexible but also more complicated:
  - How and when to detect changes
  - How to apply consistently
  - How to handle dependent/cached values
  - Do changed values need persisting?
- Testing ?

Mandatory or optional?

- Some configuration parameters must be supplied or the program cannot run
- Other values may be optional
  - a sensible default value exists
  - or less functionality is not available

Other issues

- Security
- Audit
- Upgrades
- Discoverability
- Supportability
- Manual or tool-assisted changes

Security

- Configuration data has security implications
  - Passwords (most of us expect this one!)
  - Directories
  - Paths
  - Script names
  - Database fields (SQL injection)

Security

- If it isn't configurable it is harder to hack
- Can conflict with supportability
  - (eg usually don't log the password)

Audit

- Many businesses require software audit of all changes to production systems
- How do you audit configuration changes?
  - Source code control system
    - (May want a separate repository)
  - Database
  - Manual procedures
  - Versioned file systems
- Much easier to design in than bolt on later

Upgrades

- Typically the configuration data required by a program changes during the program's lifetime
- How will you handle:
  - New items
  - Updated items
  - Deleted items?

Upgrades

- New items
  - May be able to provide a default / automatically
  - How to ensure consistency?
- Changed items
  - If old value no longer valid can cause hard to diagnose faults.
- Deleted items
  - User may expect a value has effect
  - How do you tell which data is actually in use?

Upgrades

- Rollback
  - If the upgrade is rolled back will the configuration get restored correctly?
- Sequential upgrades
  - Can you skip an upgrade?
- Decouple config change from software change

Discoverability

- What can I configure?
- What are the possible values I can use?
- How can I tell if I get it wrong?

Supportability

- If you allow configuration it *will* go wrong
- How will the program report this?
- Who will know?
- How can it be fixed?

Supportability

- How easily can you find what the current active configuration of your program *really is*?
- Can you test *just* the configuration?

Manual or tool-assisted editing?

- What mechanism is there for changing values?
- Manual editing (eg text editor, registry values)
- GUI setup page
- If both, how do you correlate them?

CROC

- In practice it's complicated
  - Mix of type of configuration items
  - Mix of static and dynamic items
  - Mix of granularity or scope (user, machine, etc)
- Unlikely that “one size fits all”

Configuration as indirection

- “All problems in computer science can be solved by another level of indirection”

(David Wheeler)

Configuration as indirection

- “All problems in computer science can be solved by another level of indirection”

```
std::string CONFIG(argc > 1 ? argv[1] : “CONFIG”);
...
if (getenv(“CONFIG”)) CONFIG = getenv(“CONFIG”);
...
sprintf(buff, “select VALUE from %s where key='CONFIG'”,
CONFIG);
…
CONFIG = select_string_value(buff);

--- environment
set CONFIG=CONFIG
```

--- database : table CONFIG:

Key	Value
CONFIG	CONFIG

Configuration as indirection

- “All problems in computer science can be solved by another level of indirection”
- … “except for the problem of too many layers of indirection” (Kevlin Henney)

Some basic patterns

- There are many patterns for configuration
- I'll look at a few and identify some of the forces and trade-offs
- Generally need to use more than one pattern

Source code

- Context
  - Value known up-front
- Benefits
  - Can be cross referenced and typed
  - Automatically audited with source code
- Liabilities
  - Produces multiple build artifacts
  - Not changed after compilation (if any...)

Source code

- Examples
  - Debug and release build
  - External programs
  - Size limits
- Factory automation example
  - Misconfiguration too expensive, so ship one file

Command line argument

- Context
  - Value known when program invoked
- Benefits
  - Easy to change manually
  - Easy to discover (on most operating systems)
- Liabilities
  - Can be hard to manage multiple items
  - 'Special characters' can be problematic
  - Hard to change programmatically
  - Audit

Command line argument

- Examples
  - Command line tools
  - Windows svchost.exe
  - Java system properties
  - Drag and drop support

Environment variables

- Context
  - Value known when program invoked
- Benefits
  - Can be set once for multiple programs
  - Easy to change
- Liabilities
  - Hard to audit and control
  - Name clashes
  - May be hard limits on sizes

Environment variables

- Examples
  - HOME
  - CLASSPATH
  - USERNAME
  - CL
- Interaction with command line adds complexity

Windows Registry

- Context
  - Windows (!)
  - Value known at program start
- Benefits
  - Standard support, e.g. by installers
  - Per user and per machine sections
  - Permissions
- Liabilities
  - Single “big ball of mud”
  - Permissions

Windows Registry

- Examples
  - COM registration
  - Installed programs
  - Policies
  - Image File Execution Options
- I'm sure we all have war stories....

Properties file (Name/Value)

- Context
  - Value available locally
- Benefits
  - Separation of concerns
  - May be able to re-write file
  - Can add comments
- Liabilites
  - Management of many small files
  - Audit
  - Restrictive syntax

Properties file

- Examples
  - Windows ini files
  - Unix rc files
  - Java properties

XML configuration file

- Context
  - Hierarchical configuration data
- Benefits
  - Flexible
  - Validated
  - Multiple tools
- Liabilities
  - Verbose
  - Not human readable
  - Different features supported (eg ENTITY)

XML configuration file

- Examples
  - app.config
  - Windows manifest files
  - Log4j configuration file
  - Spring configuration
- Pop quiz
  - which ones validate?
  - Which ones support entities?

Other file types/usages

- Unix shells sourcing files on startup
- Binary file formats used for persistence
- 'Template' pattern (using simple substitution)
- External validators

Database

- Context
  - Already *use* a database
- Benefits
  - Centralisable control and access controls
  - Range of standard data types
- Liabilities
  - Need to configure database connection details
  - Need tools to discover user's configuration

Database

- Examples
  - Relation databases often hold their own config
  - Mail servers
  - DNS lookup
  - Most programs I've worked on in recent years

External service

- Context
  - Complex or volatile configuration
- Benefits
  - Potentially more flexible than a file or database
- Liabilities
  - Need to configure the client details
  - More points of failure
  - Can be hard to test

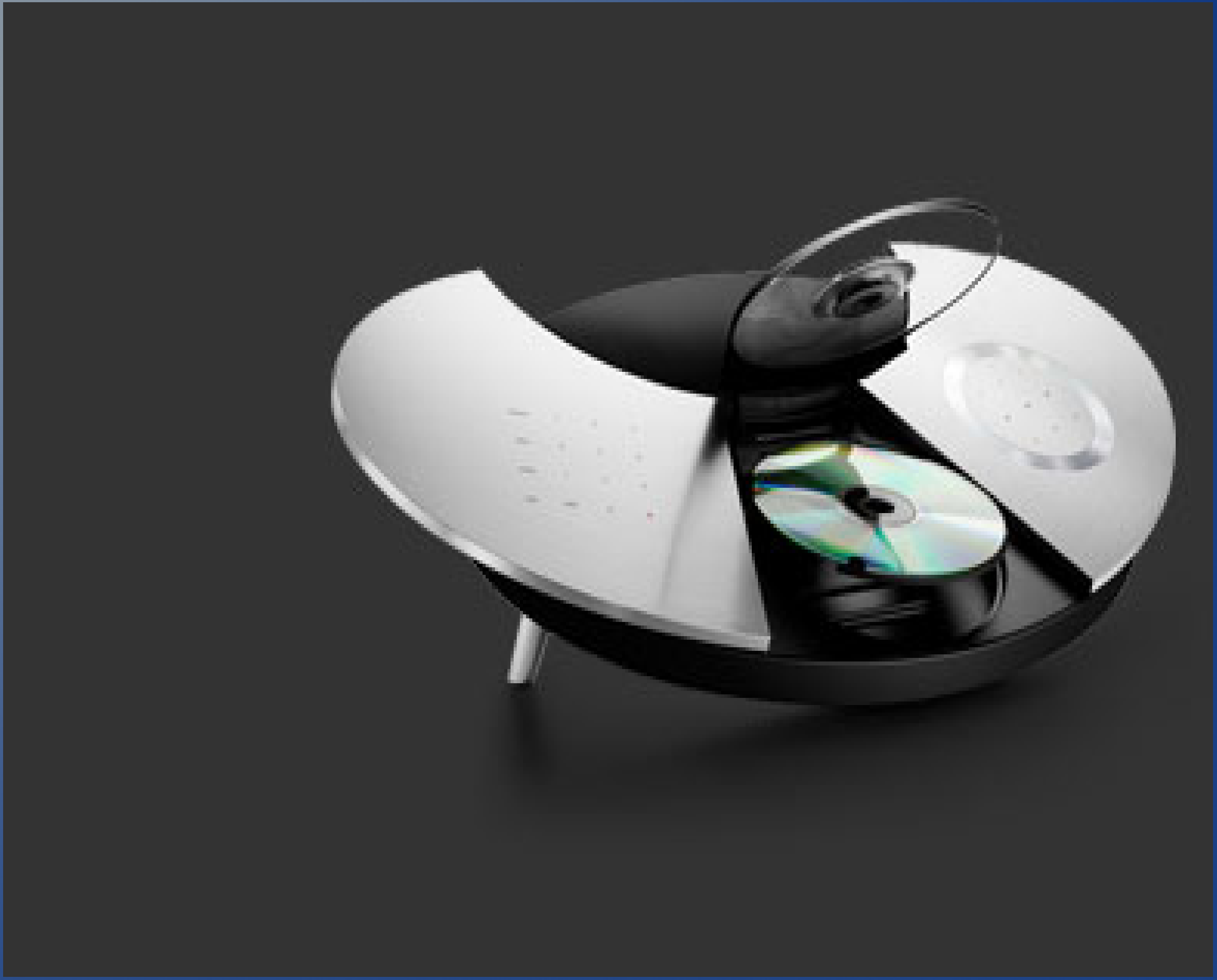
Dynamic configuration

- Polling or event driven notification?
    - Management interface
- Notifying the affected parts of the program
    - On-use
    - Callbacks
    - In-place editing
- Persisting the changed data for next time

What to do?

- Plan early for configuration
- Identify the types of configuration you need
- Eliminate unnecessary configuration
- Use smallest number of mechanisms you can

YAMAHA
01V96
STEREO



What to do?

- Support
  - Common failure modes
  - Verification
- Security
  - How can you break it?
  - What information is leaked?

Conclusion

- Configuration is often complex
- Keep the 'big picture' in mind
- Consistent project-wide configuration pays off

Conclusion

- It's not a SMOC, it's a CROC