WINGMAN SOFTWARE

SOLID C

James Grenning

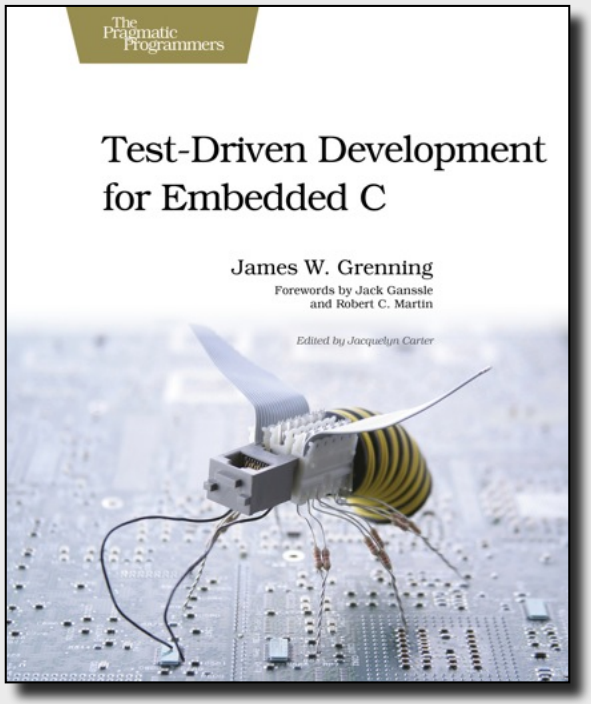
Presented at ACCU 2014

Bristol, UK

Talk to me on Twitter
@jwgrenning

Connect with me on linkedin.com
http://www.linkedin.com/in/jwgrenning
Remind me how we met.

http://www.wingman-sw.com
http://blog.wingman-sw.com
http:// www.jamesgrenning.com
unpappd.com jwgrenning

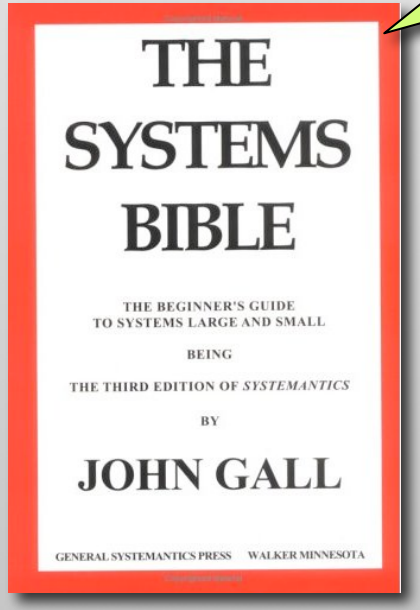


www.wingman-sw.com
james@wingman-sw.com

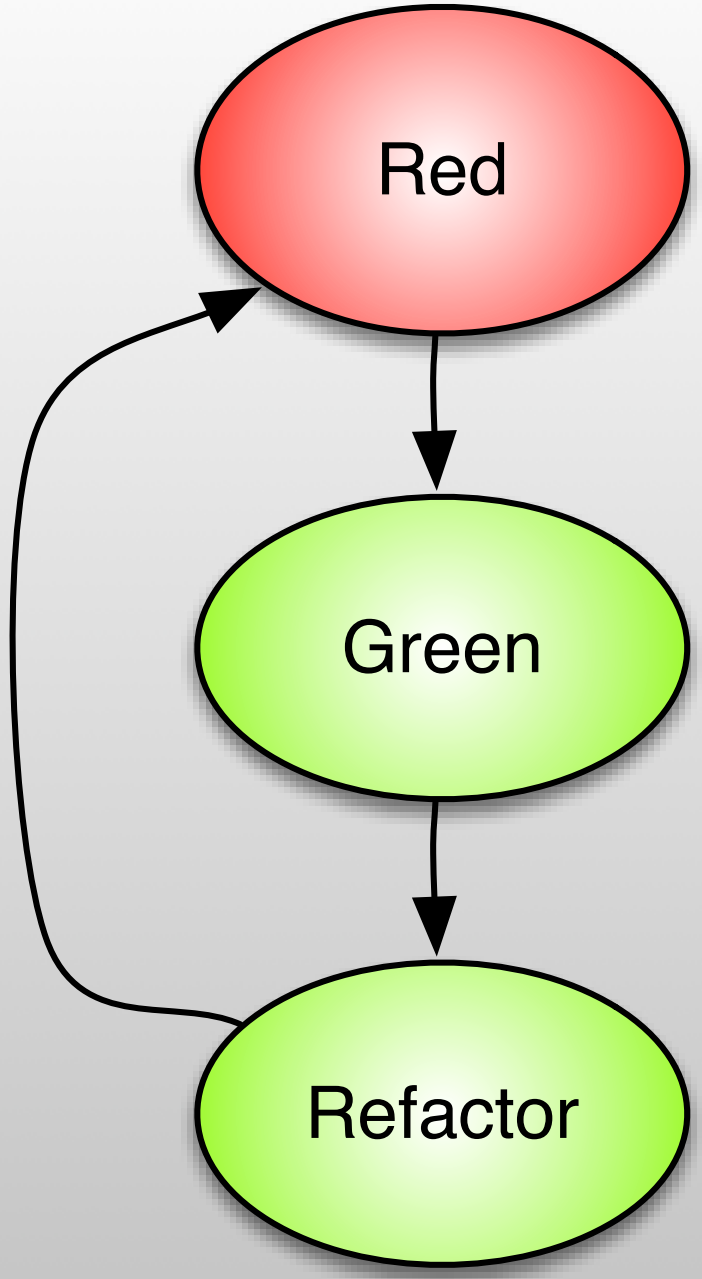
When is Software Design Finished?

www.wingman-sw.com
james@wingman-sw.com

A Complex system that works is invariably found to have evolved from a simple system that worked.



www.wingman-sw.com
james@wingman-sw.com



www.wingman-sw.com
james@wingman-sw.com

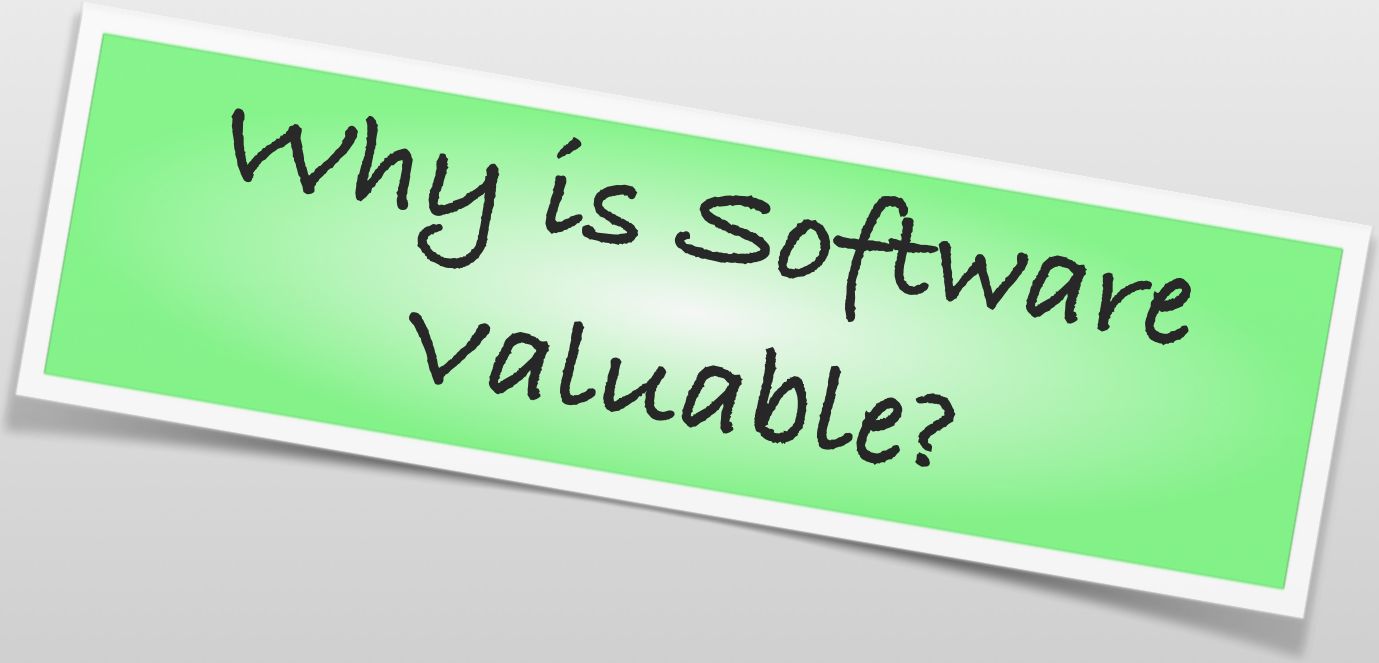
Rules of Simple Design

In Priority Order!

1. Passes all tests
2. No duplication
3. Expresses intent
4. Fewest classes and methods (no extra stuff)

Kent Beck [XP, TDD]

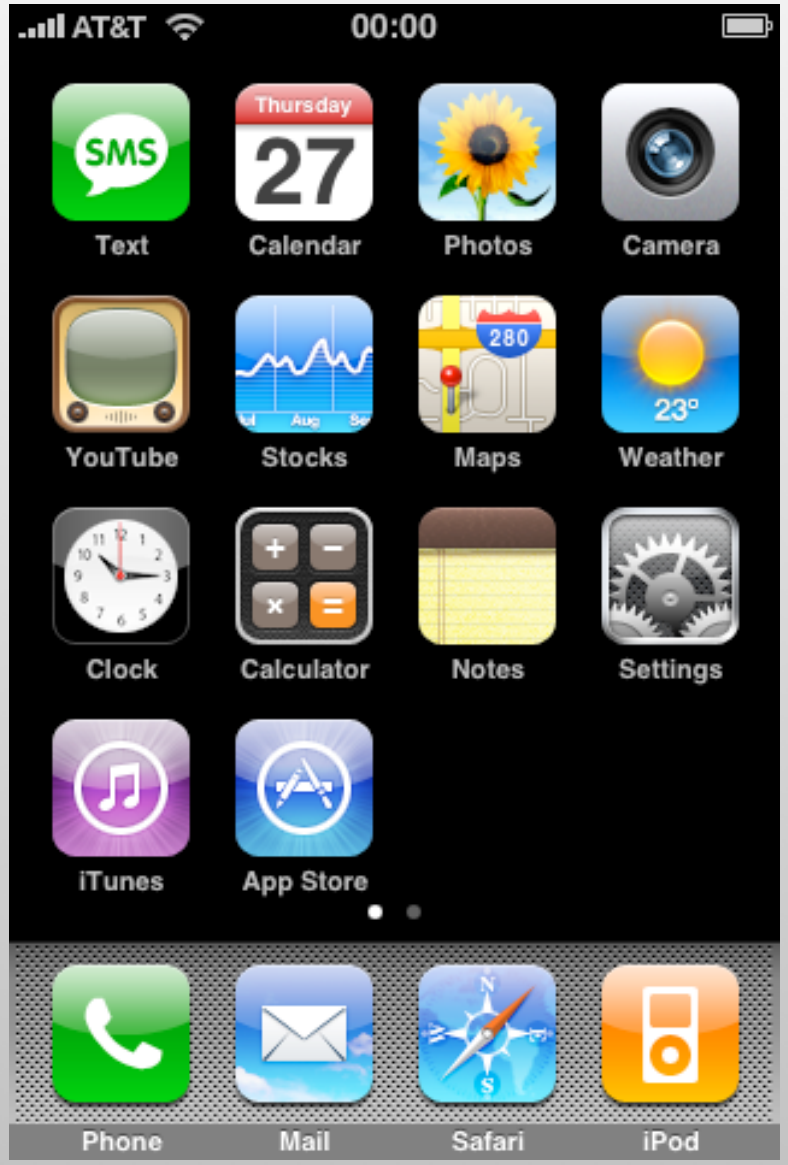
www.wingman-sw.com
james@wingman-sw.com



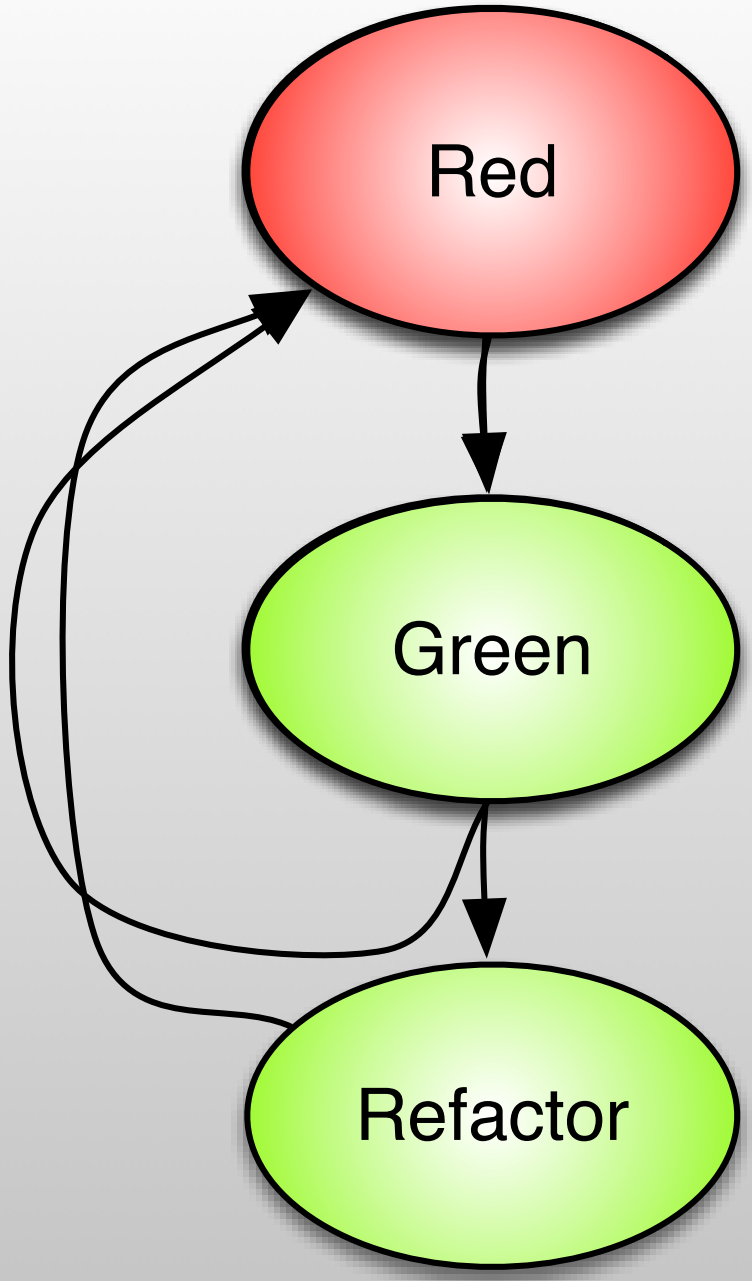
www.wingman-sw.com
james@wingman-sw.com

The Two Values of Software

- Functionality
- Malleability



www.wingman-sw.com
james@wingman-sw.com



www.wingman-sw.com
james@wingman-sw.com

The End

www.renaissancesoftware.net
james@renaissancesoftware.net

Design for Maintenance

- Systems evolve, the design is NEVER done.
- Automated tests make evolution safer.

- Key technical practices for evolving design
  - Test Driven development
  - Refactoring
  - Modularity, Loose Coupling, High Cohesion (OO Design)

www.wingman-sw.com
james@wingman-sw.com

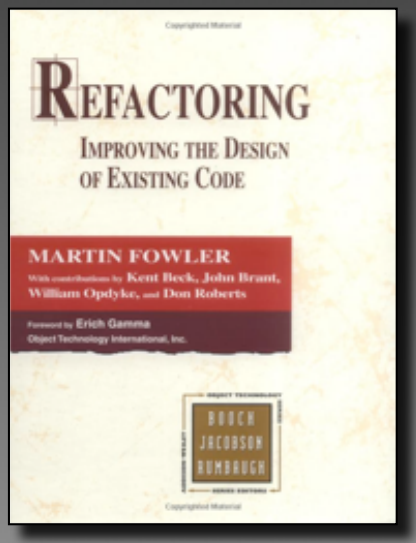

Donald Knuth Says

Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

www.wingman-sw.com
james@wingman-sw.com

Martin Fowler Says

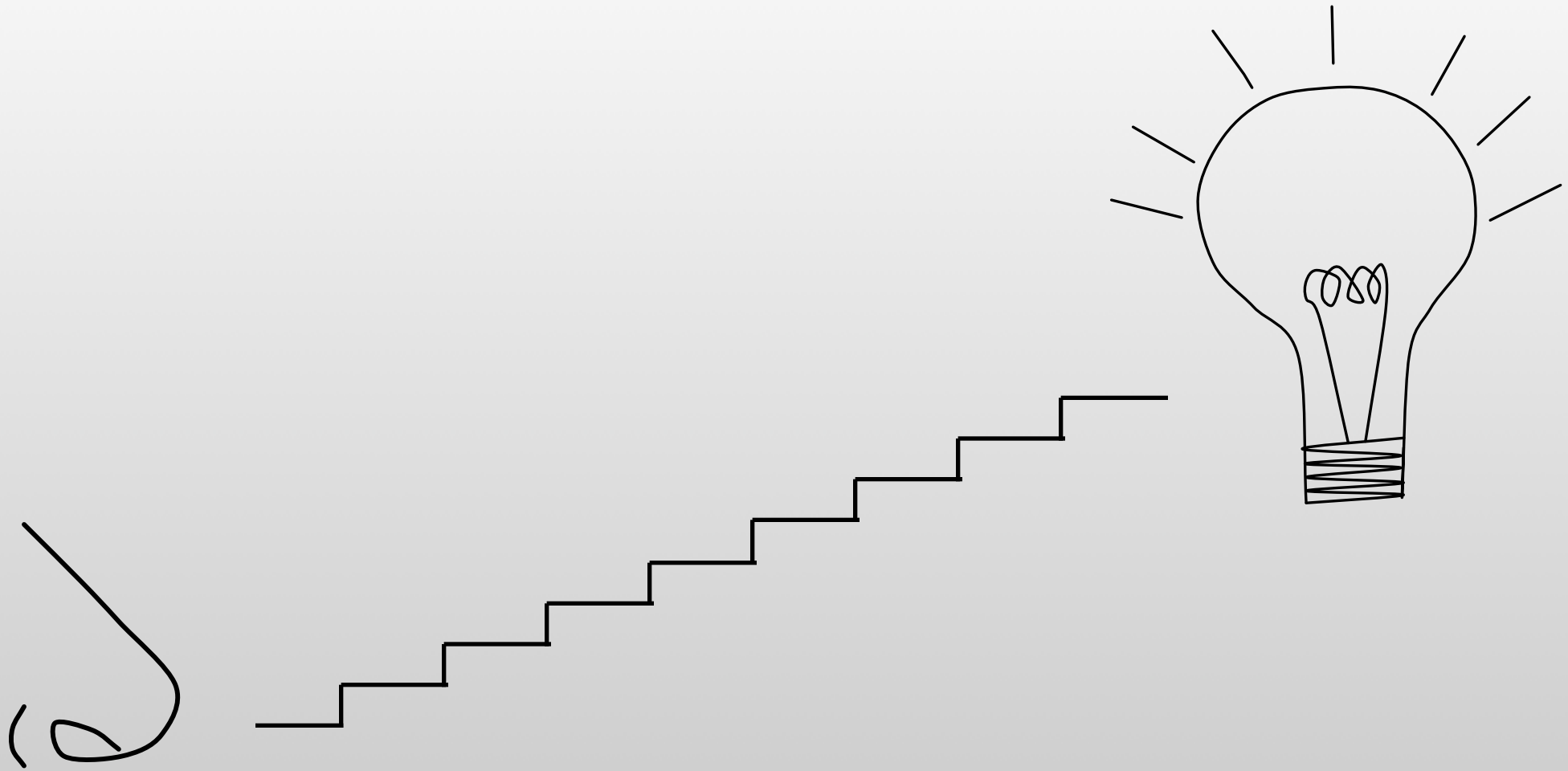
Any fool can write code that the compiler understands, but it takes real skill to write code other programmers can understand.



From "Refactoring - Improving the Design of Existing Code

www.wingman-sw.com
james@wingman-sw.com

Three Critical Skills



Recognize what is wrong and fix it!

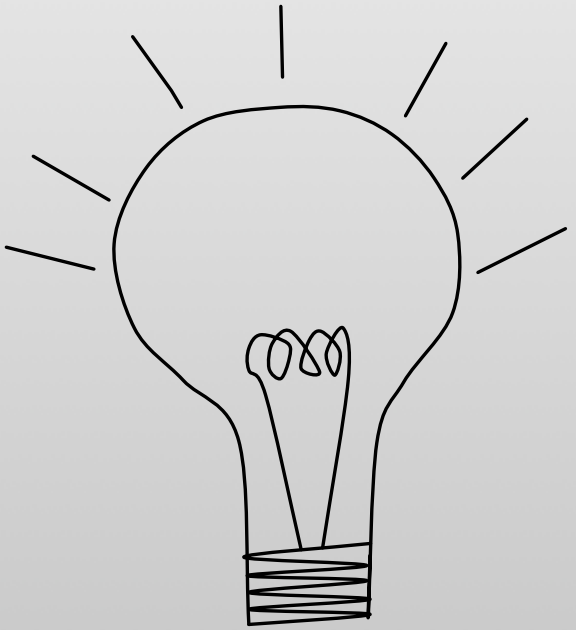
www.wingman-sw.com
james@wingman-sw.com

SOLID

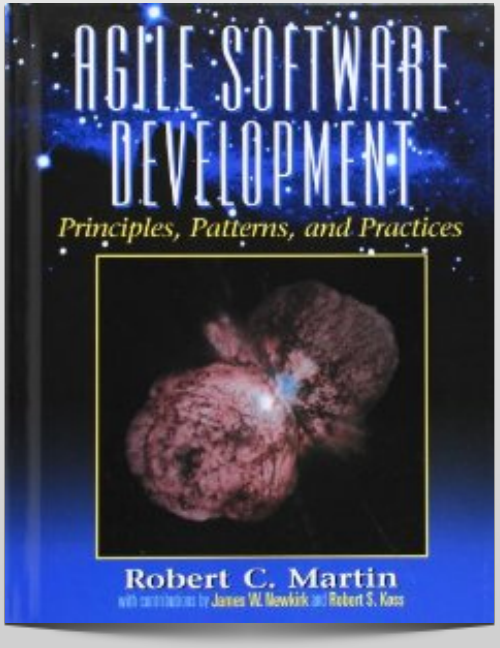
www.wingman-sw.com
james@wingman-sw.com

Envisioning

- Once a design problem is identified, you must envision a better solution
- Look to apply the design principles
  - SOLID
  - DRY - Don't Repeat Yourself
    - from The Pragmatic Programmer
  - Principle of Least Knowledge.
  - Separation of Concerns (SoC).



SOLID Design Principles



www.wingman-sw.com
james@wingman-sw.com

SOLID Design Principles

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

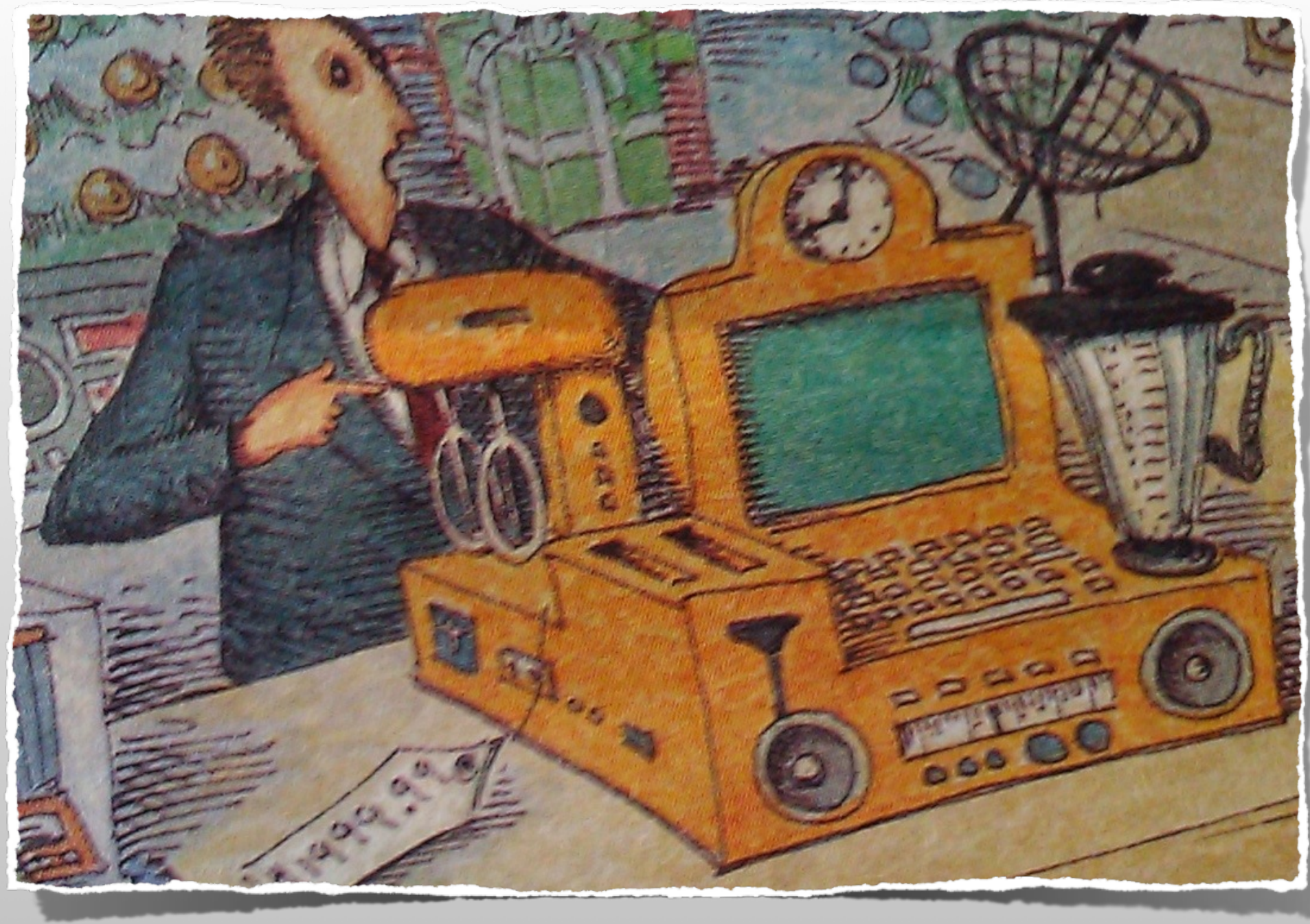
Interface Segregation Principle

Dependency Inversion Principle

www.wingman-sw.com
james@wingman-sw.com

Home Automation System
Light Scheduling Requirements

- The light scheduler maintains a schedule for lights that can be turned on or off. Multiple schedules per light are supported for each light.
- A light can be scheduled to turn on.
- A light can be scheduled to turn off.
- 32 lights can be addressed by their integer ID.
- It must support 128 separate scheduled events
- Schedules can be established by
  - specific day of the week (M Tu W Th F Sa Su), or everyday.
- Lights can also be controlled though a UI on a front panel, iPad, iPhone, Android device, or web enabled device.



www.wingman-sw.com
james@wingman-sw.com

Single Responsibility Principle

Bob Martin [AGILE]

- a.k.a. Cohesion

- A module should do one thing and do it well.
- A module should have a single reason to change.

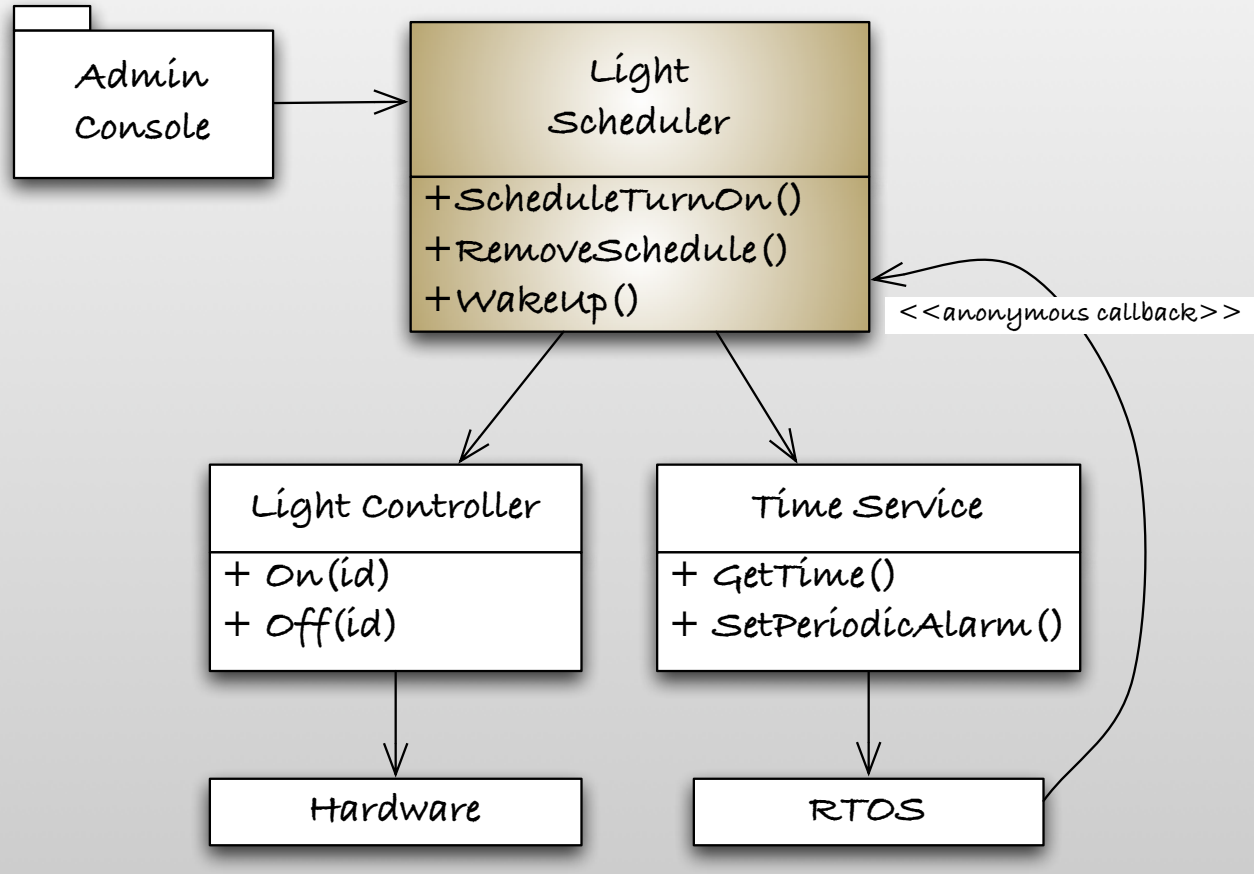
- Test: a module can be described in 25 words or less.

www.wingman-sw.com
james@wingman-sw.com


james@wingman-sw.com

A Module with Collaborators

- Every minute, the RTOS wakes up the Light Scheduler.
- If it is time for one of the lights to be controlled, the LightController is told to turn on/off the light.



www.wingman-sw.com
james@wingman-sw.com

Open/Closed Principle

- A module is open for extension, and closed for modification.
- A design adheres to the open closed principle when it accommodates a certain kinds of changes without having to change existing code.
- Changes just drop in.

- Say "X is open for extension for new kinds of Ys, but closed for modification"

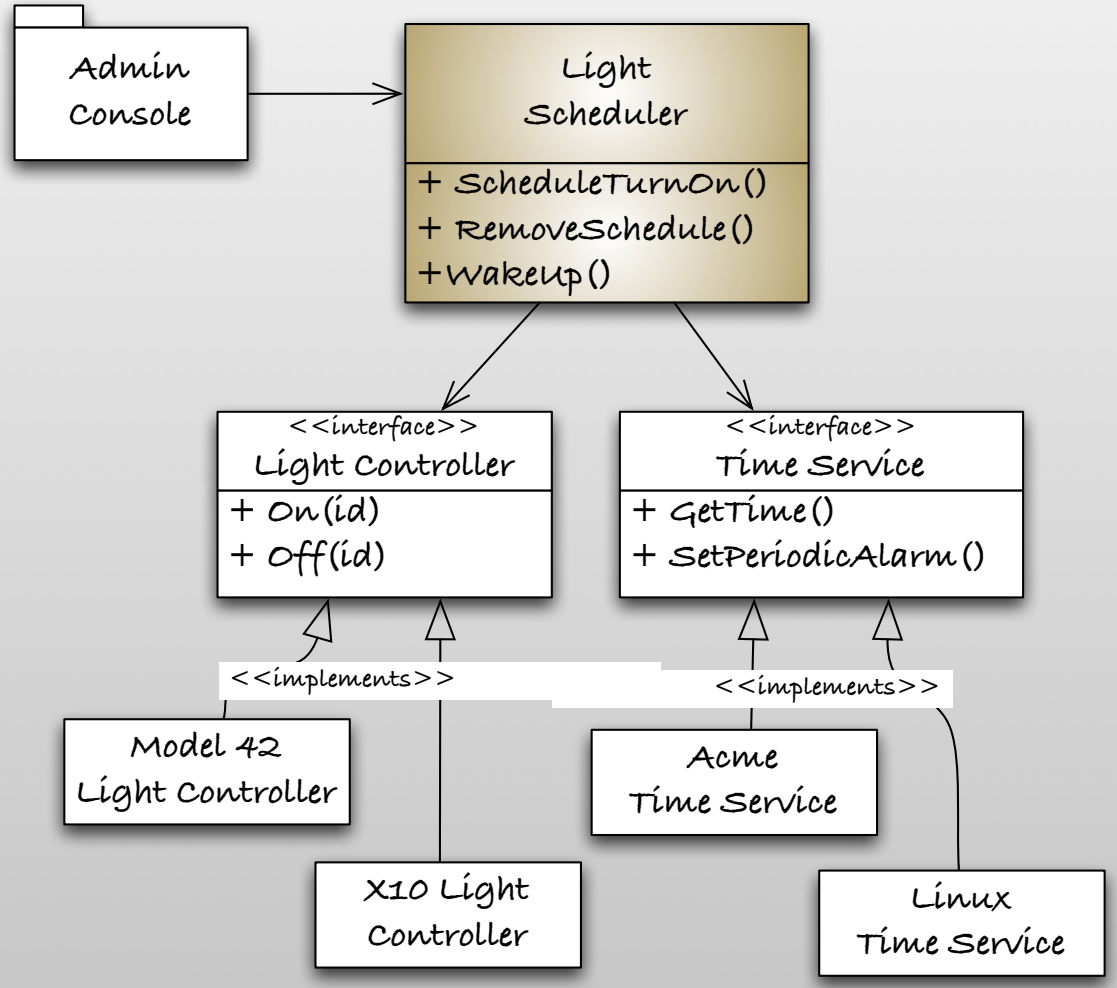
Thanks: Bertrand Meyers

www.wingman-sw.com
james@wingman-sw.com

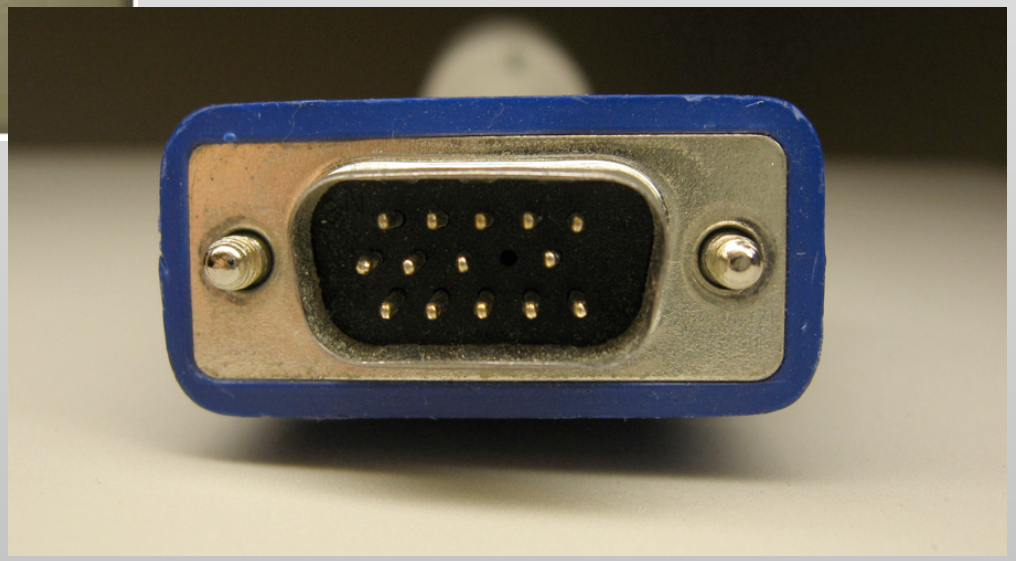

OCP Applied

LightScheduler is **open** for extension for new kinds of LightControllers and TimeServices.

It is **closed** for modification for those extensions.



www.wingman-sw.com
james@wingman-sw.com



www.wingman-sw.com
james@wingman-sw.com

Liskov Substitution Principle

- Modules with the same interface can be substituted without the client knowing the difference.
  - Modules must meet the contract of the client and interface.
  - Preconditions cannot be strengthened in a server.
  - Postconditions cannot be weakened in a server.

Thanks: Barbara Liskov

www.wingman-sw.com
james@wingman-sw.com


OS Abstraction

```
typedef struct ThreadStruct * Thread;
typedef void * (*ThreadEntryFunction)(void *);

Thread Thread_Create(ThreadEntryFunction f, void * parameter);
void Thread_Start(Thread);
void Thread_Destroy(Thread);
void Thread_Exit(void *);
void Thread_Join(Thread, void **result);
void * Thread_Result(Thread);
```


www.wingman-sw.com
james@wingman-sw.com

OS Abstraction is Not Working

```
   //Snip from some product code

Thread thread = Thread_Create(threadEntry, 0);

#if POSIX_OS
    //POSIX starts thread on create
#else
    //AcmeOS does not start thread on create
    Thread_Start(thread);
#endif
```


www.wingman-sw.com
james@wingman-sw.com

OCP/LSP Designs Don't Burden the Client

```
//Snip from some product code

Thread thread = Thread_Create(threadEntry, 0);
Thread_Start(thread);
```


www.wingman-sw.com
james@wingman-sw.com

POSIX Version Delays Thread Creation

```
typedef struct ThreadStruct
{
    ThreadEntryFunction entry;
    void * parameter;
    pthread_t pthread;
    BOOL started;

} ThreadStruct;

Thread Thread_Create(ThreadEntryFunction f, void * parameter)
{
     Thread self = calloc(1, sizeof(ThreadStruct));
     self->entry = f;
     self->parameter = parameter;
     self->started = FALSE;;
     return self;
}

void Thread_Start(Thread self)
{
    self->started = TRUE;
    pthread_create(&self->pthread, NULL, self->entry, self->parameter);
}
```


www.wingman-sw.com
james@wingman-sw.com


Acme Does Not Need To Delay Create

```
typedef struct ThreadStruct
{
    AcmeThreadStruct acmeThread;
} ThreadStruct;

Thread Thread_Create(ThreadEntryFunction entry, void * parameter)
{
     Thread self = calloc(1, sizeof(ThreadStruct));
     AcmeThread_create(&self->acmeThread, entry, parameter, 5, 1000);
     return self;
}

void Thread_Start(Thread self)
{
    AcmeThread_start(&self->acmeThread);
}
```

Interface Segregation Principle

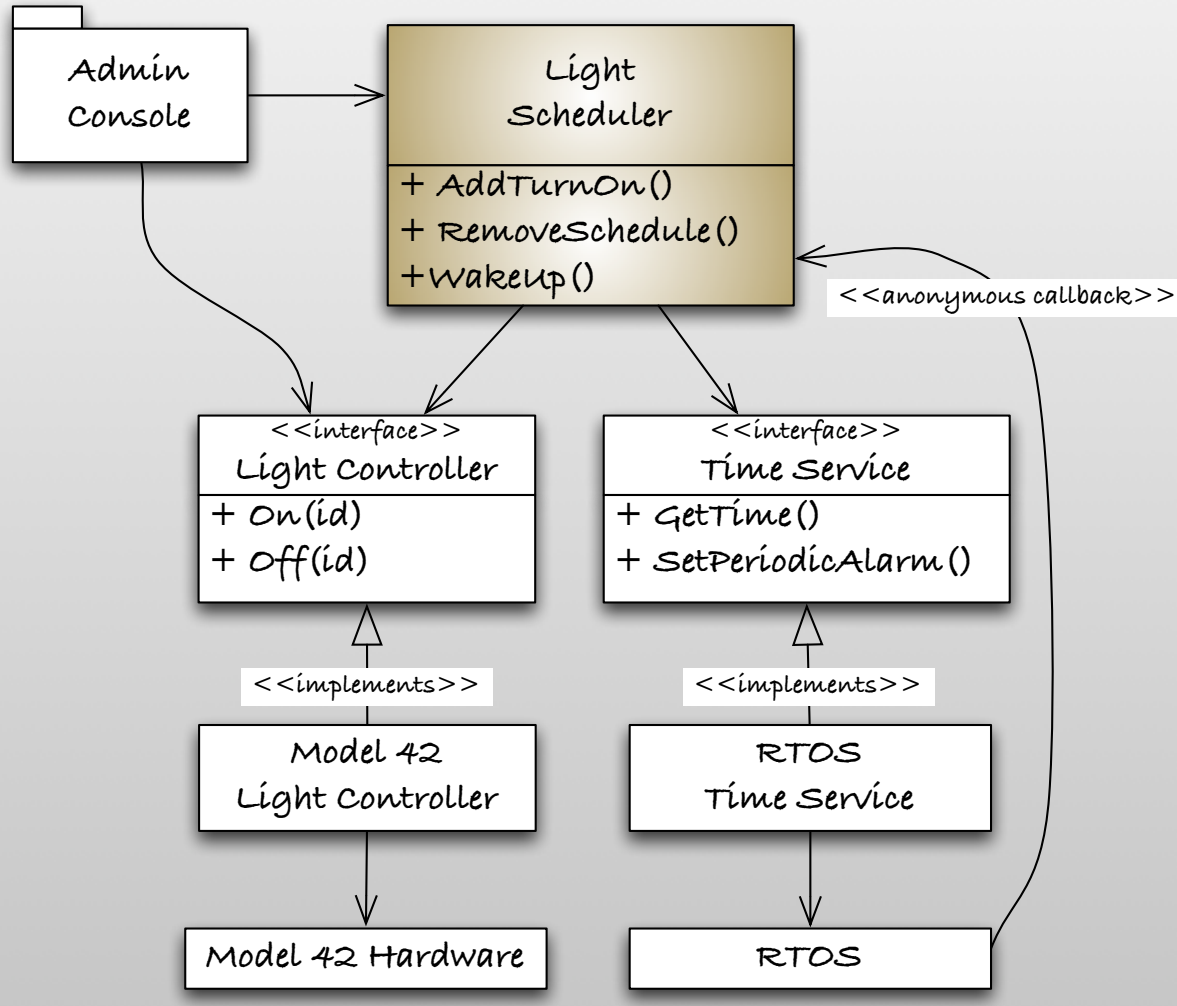
- Don't depend on fat interfaces.
- Don't depend on interfaces that have methods you don't care about.
- Tailor interfaces to client need.
- Split fat interfaces.

Thanks: Robert Martin

www.wingman-sw.com
james@wingman-sw.com

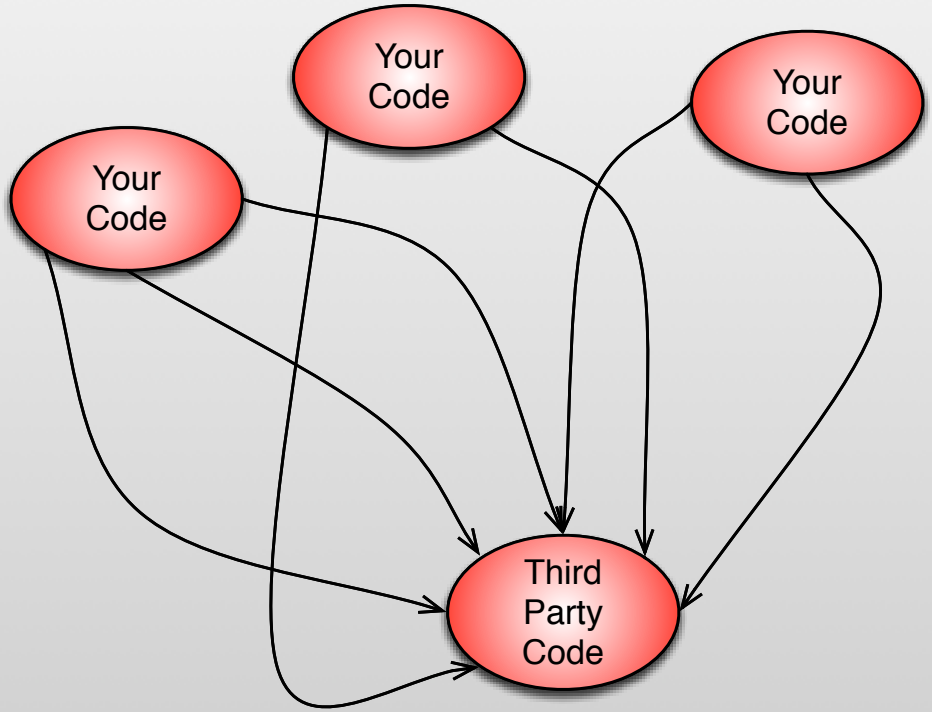
Light Scheduler Design

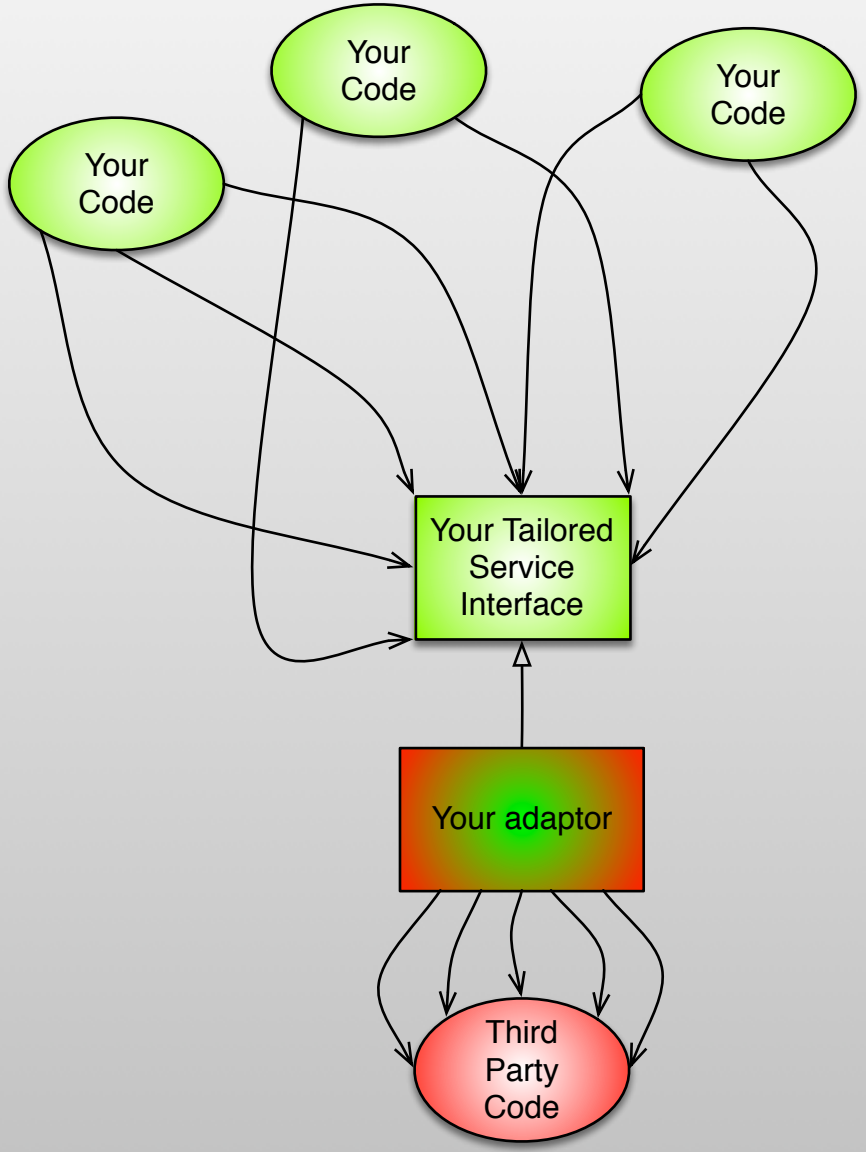
- The TimeService limits the knowledge of the RTOS
- Adapters are a form of Interface Segregations



www.wingman-sw.com
james@wingman-sw.com

Manage Third Party Dependenices





www.wingman-sw.com
james@wingman-sw.com

Dependency Inversion Principle

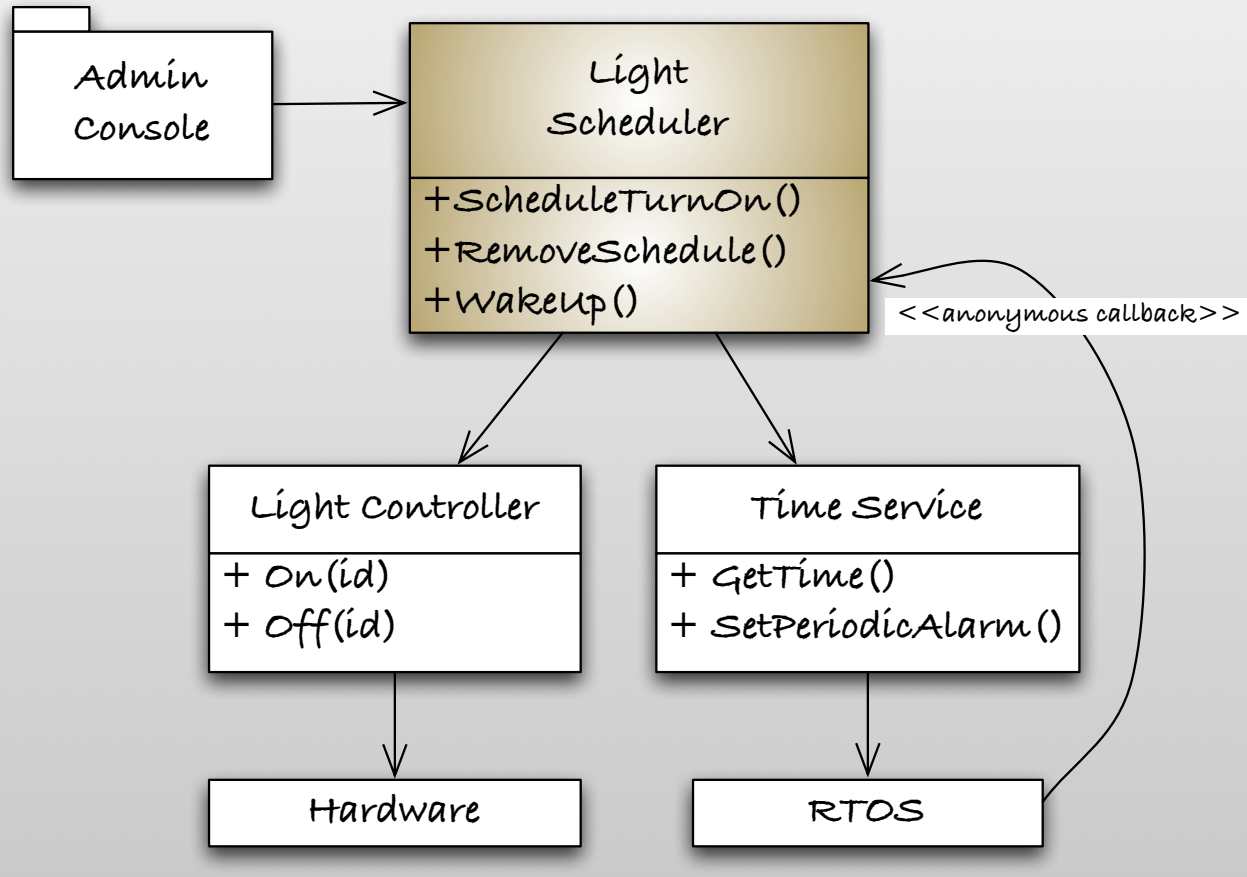
- Invert dependencies from high level modules to low level details.
- Break problem dependencies by inserting an abstraction.
  - High level detail depends on an interface
  - Low level code depends on the interface
  - Interface depends on neither
  - Dependency cycle
- a.k.a. Depend on Abstractions

Thanks: Robert Martin

www.wingman-sw.com
james@wingman-sw.com

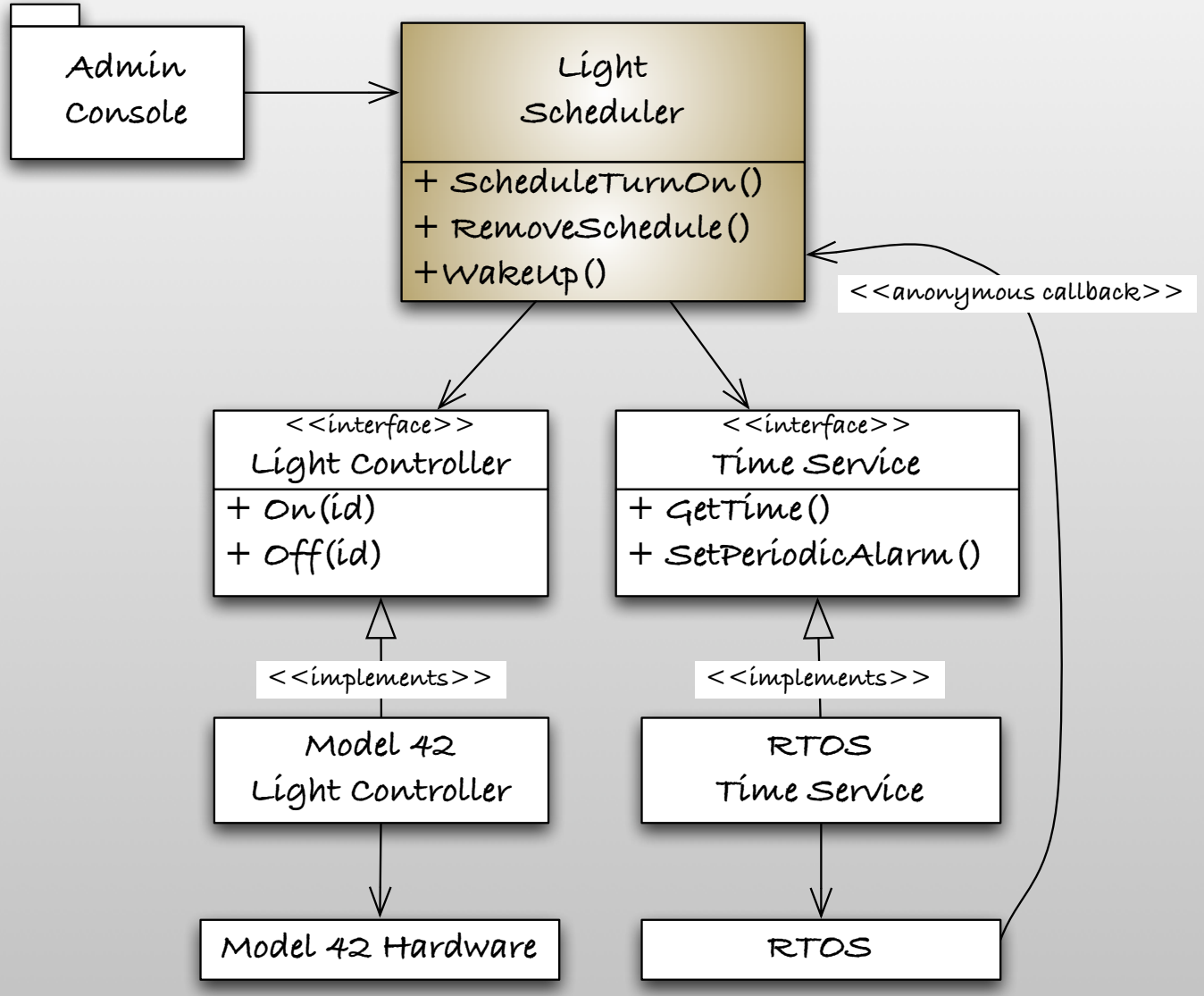

Un-Managed Dependencies

- Transitive dependencies make the highest level of this design depend on the lowest level details.
- Use DIP to break the transitive dependency chain.



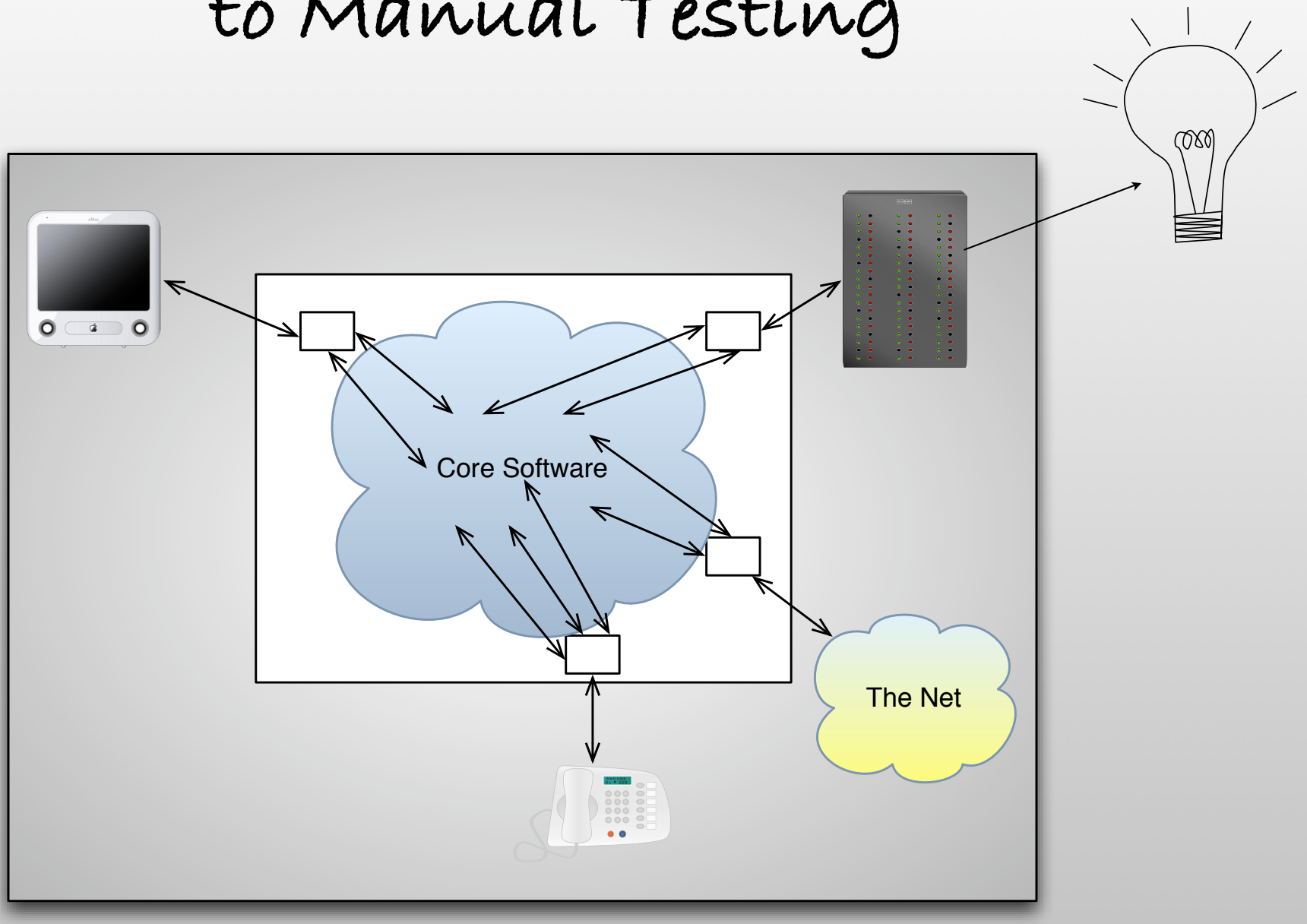
www.wingman-sw.com
james@wingman-sw.com

Invert Dependencies with Interfaces



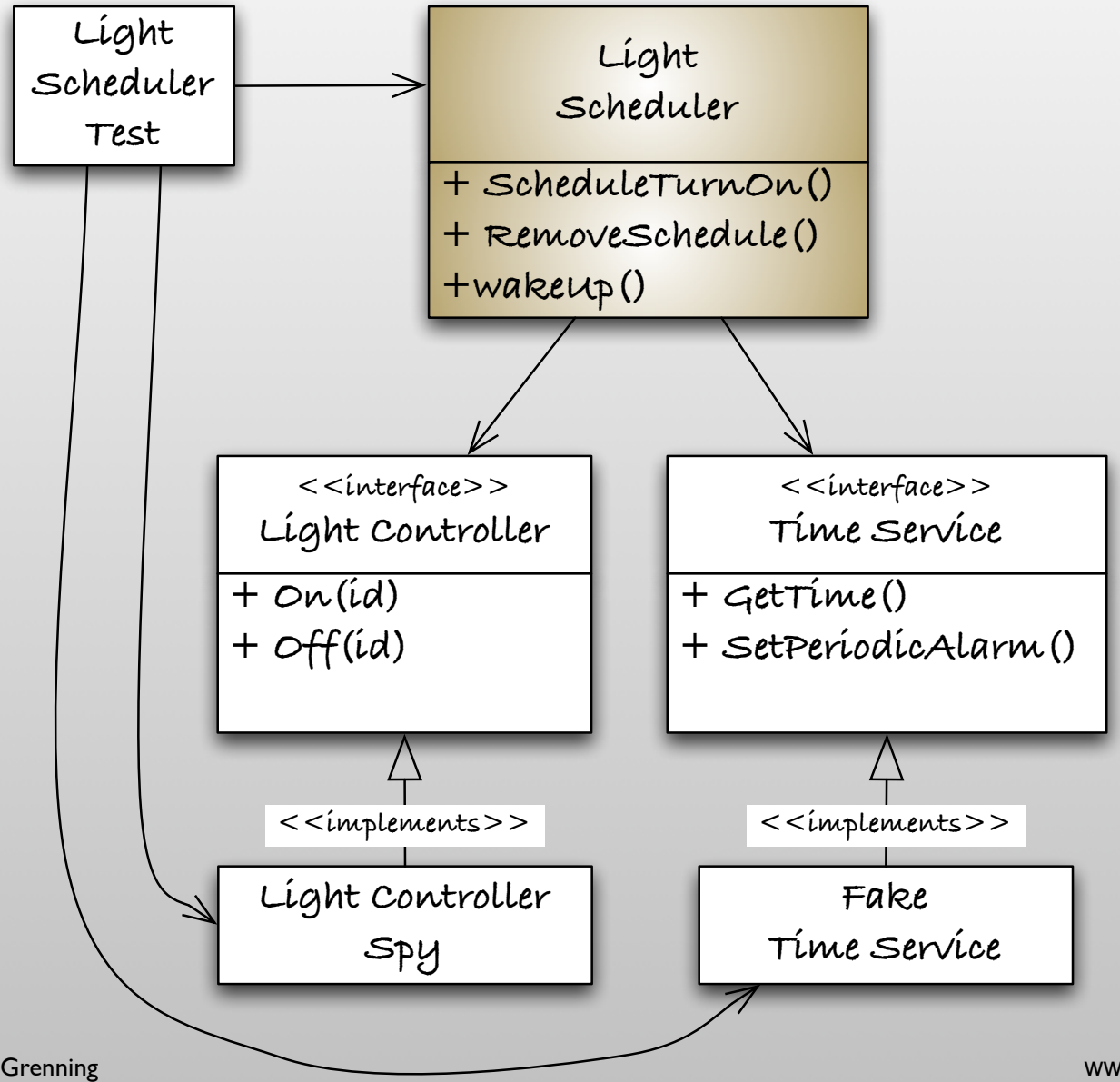
www.wingman-sw.com
james@wingman-sw.com

Unmanaged Dependencies Lead to Manual Testing



www.wingman-sw.com
james@wingman-sw.com

Substitutability Supports Testing



www.wingman-sw.com
james@wingman-sw.com

Interfaces in C

- Containing just what is needed to interact with the module
  - Function declarations
  - Constants needed for interacting with the module
  - struct forward declaration

- Things not in the interface header file
  - Data structure member definitions
  - Data definitions
  - Constants needed in the implementation

Rules of Simple Design

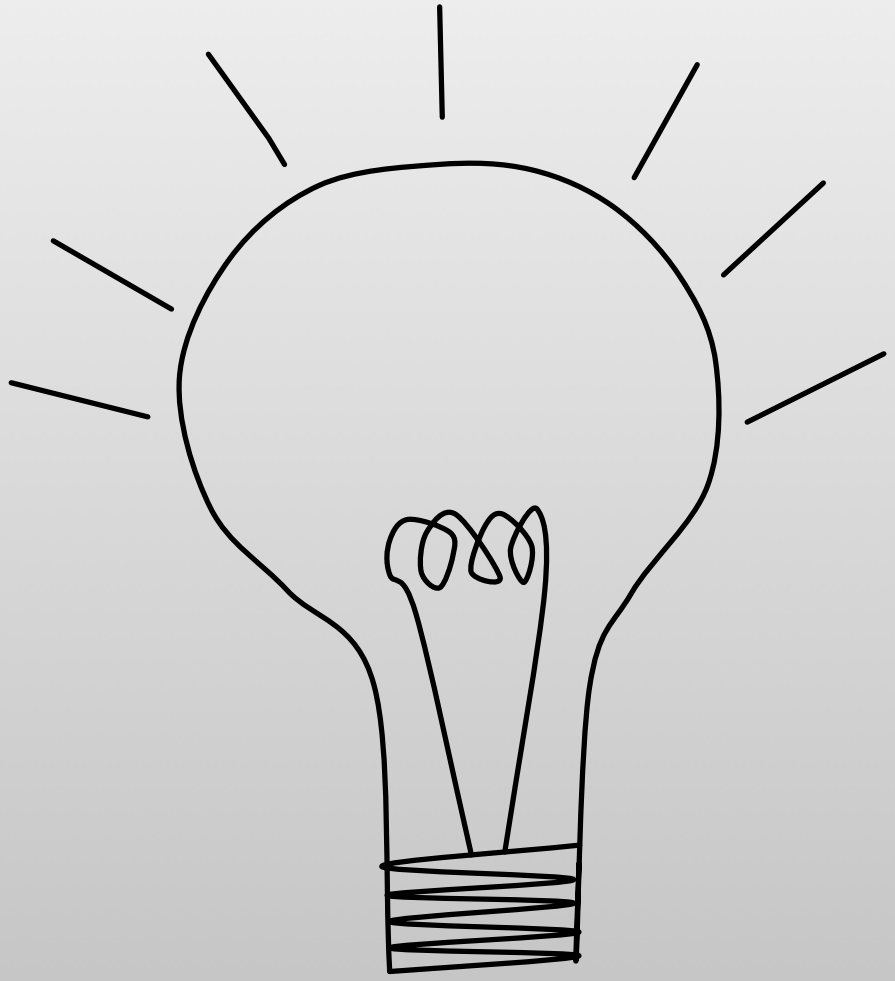
In Priority Order!

1. Passes all tests
2. No duplication
3. Expresses intent
4. Fewest classes and methods (no extra stuff)*

Kent Beck [XP, TDD]

www.wingman-sw.com
james@wingman-sw.com

*Fewest Classes and Methods that Move You Closer to A Workable Architecture



www.wingman-sw.com
james@wingman-sw.com

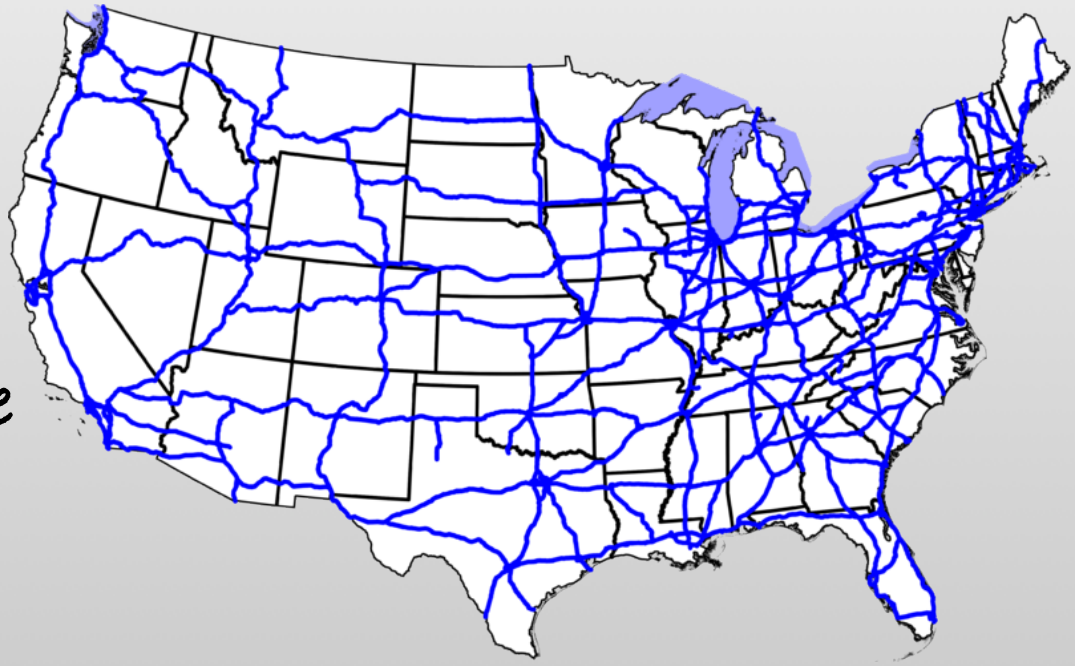
Architectural Vision

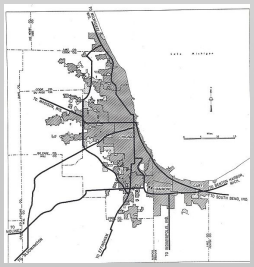
- Have an idea where you are going.
- Build slices of functionality that lets you try the architecture.
- Don't build architecture first, get features working and refactor to a great architecture.
- Keep architectural view high level.

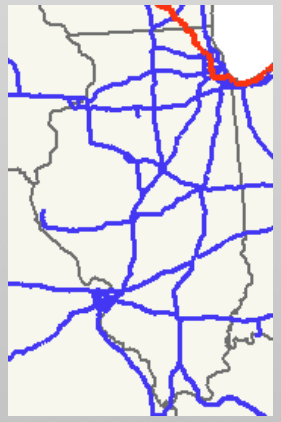
www.wingman-sw.com
james@wingman-sw.com


Architectural Vision
The Big Picture

- Have a big picture in the developers' minds.
- The vision shows
  - the major boundaries
  - the areas of concern
  - the interfaces
- Likely fractal in nature







www.wingman-sw.com
james@wingman-sw.com

SOLID and Testable C

www.wingman-sw.com
james@wingman-sw.com

Different Design Models in C

Model	Purpose
Single-instance Abstract Data Type	Encapsulates a module's internal state when only a single instance of the module is needed
Multiple-instance Abstract Data Type	Encapsulates a module's internal state and allows multiple instances of the module's data
Dynamic interface	Allows a module's interface functions to be assigned at runtime
Per-type dynamic Interface	Allows multiple types of modules with the same interface to have unique interface functions

Barbara Liskov,
Programming with abstract data types. Proceedings of the
ACM SIGPLAN Symposium on Very High Level Languages, 1974.

www.wingman-sw.com
james@wingman-sw.com


Single Instance Abstract Data Type

```
#ifndef D_LightScheduler_H
#define D_LightScheduler_H

#include "TimeService.h"

enum Day {
    NONE=-1, EVERYDAY=10, WEEKDAY, WEEKEND,
    SUNDAY=1, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
};

typedef enum Day Day;

void LightScheduler_Create();
void LightScheduler_Destroy();
void LightScheduler_ScheduleTurnOn(int id, Day day, int minute);
void LightScheduler_ScheduleTurnOff(int id, Day day, int minute);
void LightScheduler_ScheduleRemove(int id, Day day, int minute);
void LightScheduler_Wakeup(Time*);

#endif  // D_LightScheduler_H
```

Data hidden in C file using file scope

www.wingman-sw.com
james@wingman-sw.com

Multiple Instance Abstract Data Type Clients Only Interact Through the Interface

```
#ifndef D_CircularBuffer_H
#define D_CircularBuffer_H

typedef struct CircularBuffer CircularBuffer;

CircularBuffer * CircularBuffer_Create(int capacity);
void CircularBuffer_Destroy(CircularBuffer);
int CircularBuffer_IsEmpty(CircularBuffer);
int CircularBuffer_IsFull(CircularBuffer);
int CircularBuffer_Put(CircularBuffer, int);
int CircularBuffer_Get(CircularBuffer);
int CircularBuffer_Capacity(CircularBuffer);
#endif  // D_CircularBuffer_H
```

Public name, hidden internals

instance returned

instance passed in

www.wingman-sw.com
james@wingman-sw.com

Multiple Instance ADT - .c File Structure Fields are Private

```
#include "CircularBuffer.h"
#include "Utils.h"
#include <stdlib.h>
#include <string.h>

typedef struct CircularBuffer
{
    int count;
    int index;
    int outdex;
    int capacity;
    int * values;
} CircularBuffer;
```


www.wingman-sw.com
james@wingman-sw.com

Multiple Instance ADT - .c File Well Defined Initialization and Cleanup

```
CircularBuffer * CircularBuffer_Create(int capacity)
{
    CircularBuffer * self = calloc(capacity,
                                   sizeof(CircularBuffer));
    self->capacity = capacity;
    self->values = calloc(capacity + 1, sizeof(int));
    self->values[capacity] = BUFFER_GUARD;
    return self;
}

void CircularBuffer_Destroy(CircularBuffer * self)
{
    free(self->values);
    free(self);
}
```

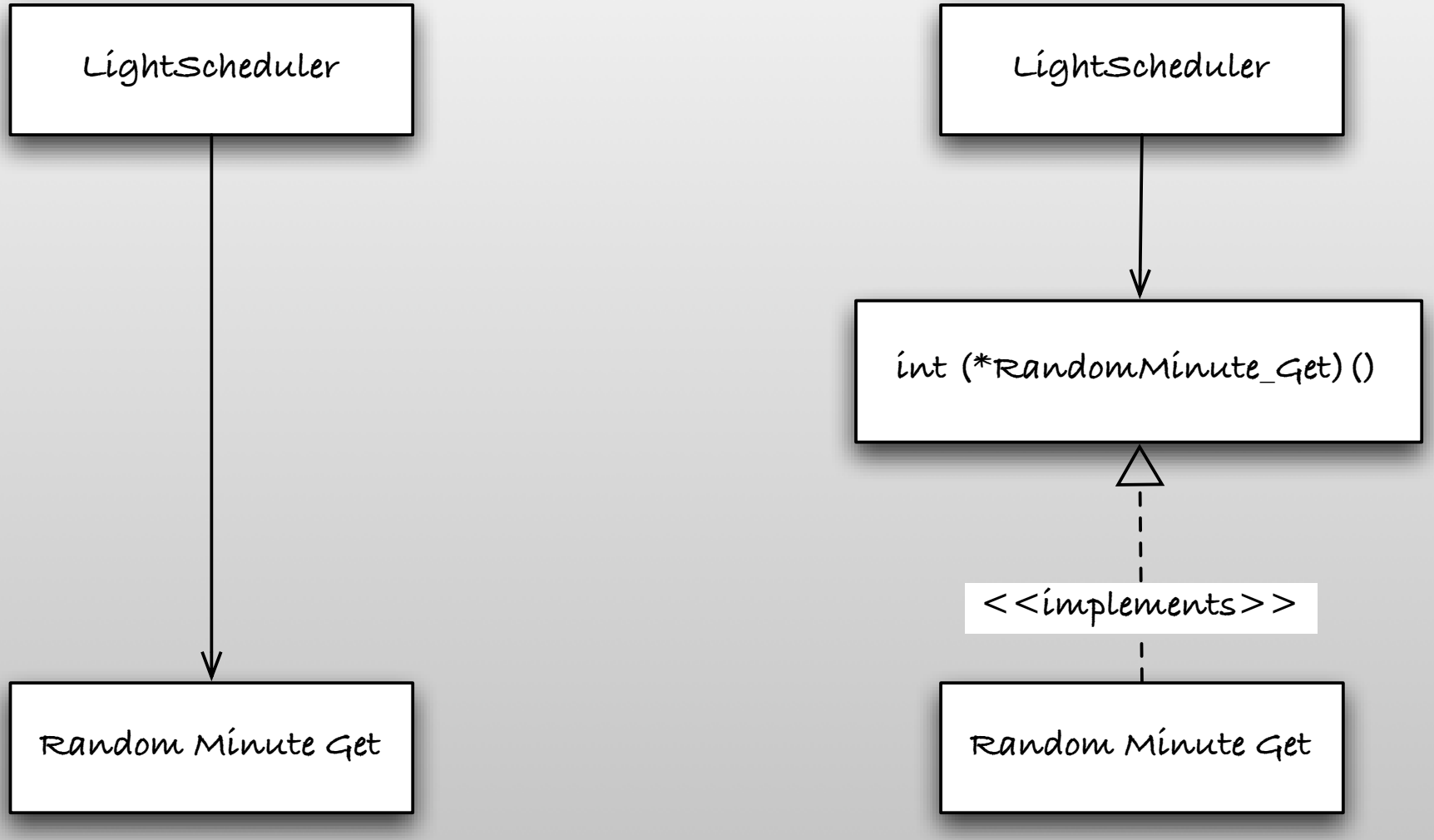
Simple Dynamic Implementation

```
#ifndef D_RandomMinuteGenerator_H
#define D_RandomMinuteGenerator_H

void RandomMinuteGenerator_Create(int bound);
extern int (*RandomMinuteGenerator_Get)();

#endif  // D_RandomMinuteGenerator_H
```

function pointer interface

www.wingman-sw.com
james@wingman-sw.com

Inverting a Dependency with a Function Pointer



www.wingman-sw.com
james@wingman-sw.com

#if Abuse

```
void LightController_TurnOn(int id)
{
    LightDriver driver = lightDrivers[id];
    if (NULL == driver) return;

    #if X10_LIGHTS
        X10LightDriver_TurnOn(driver);
    #elif ACME_LIGHTS
       AcmeWirelessLightDriver_TurnOn(driver);
    #elif MEMORY_MAPPED_LIGHTS
       MemMappedLightDriver_TurnOn(driver);
    #elif TESTING
       LightDriverSpy_TurnOn(driver);
    #endif
}
```

Similar **#if** statements litter the code. This does not look too DRY

www.wingman-sw.com
james@wingman-sw.com

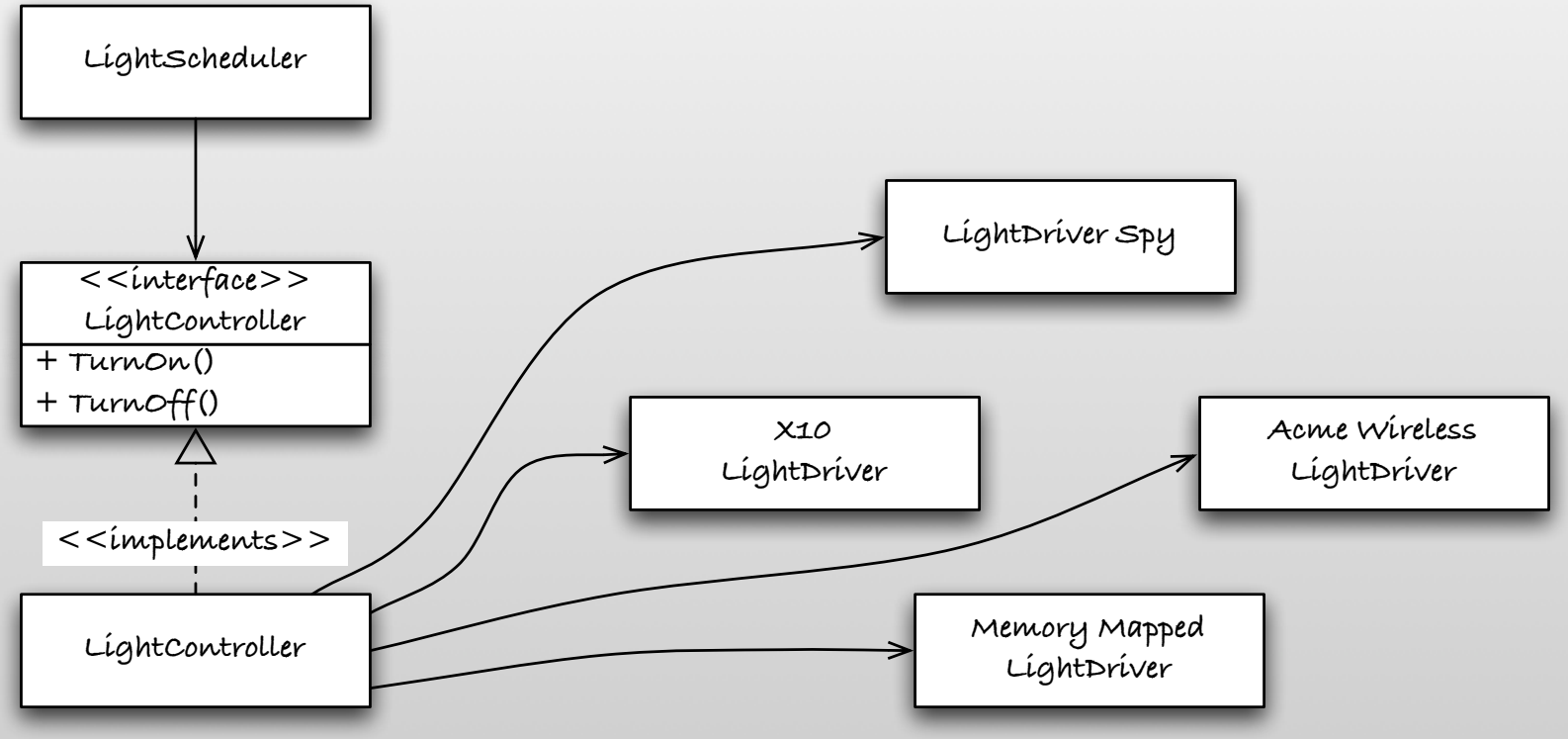
switch case Abuse

```
void LightController_TurnOn(int id)
{
    LightDriver driver = lightDrivers[id];
    if (NULL == driver) return;

    switch (driver->type)
    {
    case X10:
        X10LightDriver_TurnOn(driver);
        break;
    case AcmeWireless:
        AcmeWirelessLightDriver_TurnOn(driver);
        break;
    case MemoryMapped:
        MemMappedLightDriver_TurnOn(driver);
        break;
    case TestLightDriver:
        LightDriverSpy_TurnOn(driver);
        break;
    default:
        /* now what? */
        break;
    }
}
```

Similar **switch cases** litter the code when switching on type

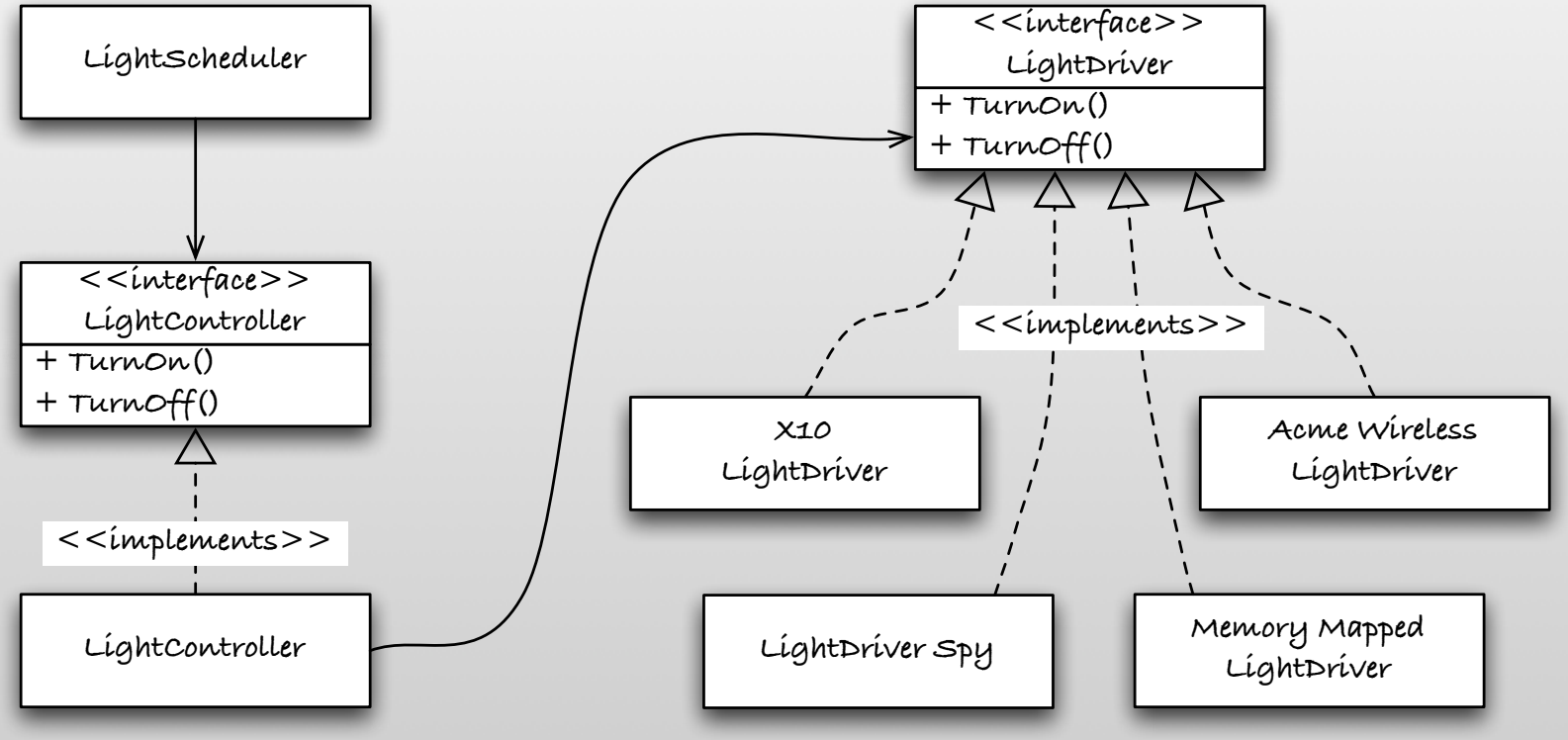
www.wingman-sw.com
james@wingman-sw.com

LightController depends on Each Specific LightDriver



www.wingman-sw.com
james@wingman-sw.com

LightController Depends on the LightDriver Interface



www.wingman-sw.com
james@wingman-sw.com

How Much Flexibility is Needed?

- Support for one LightDriver at compile time?
  - use the linker
- Support for one LightDriver type determined at runtime?
  - use a function pointer interface
- Support for multiple LightDriver types determined at runtime?
  - use per-type dynamic interface

LightDriver.h

```
typedef struct LightDriverStruct  * LightDriver;

void LightDriver_Destroy(LightDriver);
void LightDriver_TurnOn(LightDriver);
void LightDriver_TurnOff(LightDriver);
const char * LightDriver_GetType(LightDriver driver);
int LightDriver_GetId(LightDriver driver);


#include "LightDriverPrivate.h"
```


www.wingman-sw.com
james@wingman-sw.com

LightDriverPrivate.h

```
typedef struct LightDriverInterfaceStruct *
LightDriverInterface;

typedef struct LightDriverStruct
{
    LightDriverInterface vtable;
    const char * type;
    int id;
} LightDriverStruct;

typedef struct LightDriverInterfaceStruct
{
    void (*TurnOn)(LightDriver);
    void (*TurnOff)(LightDriver);
    void (*Destroy)(LightDriver);
} LightDriverInterfaceStruct;
```

function pointer table

www.wingman-sw.com
james@wingman-sw.com

X10LightDriver.h

```
#include "LightDriver.h"

typedef struct X10LightDriverStruct * X10LightDriver;

typedef enum X10_HouseCode {
    X10_A,X10_B,X10_C,X10_D,X10_E,X10_F,
    X10_G,X10_H,X10_I,X10_J,X10_K,X10_L,
    X10_M,X10_N,X10_O,X10_P } X10_HouseCode;

LightDriver X10LightDriver_Create(int id,
                                  X10_HouseCode code,
                                  int unit);
```

Extending LightDriver struct
(X10LightDriver.c)

```
//snip...

typedef struct X10LightDriverStruct
{
    LightDriverStruct base;
    X10_HouseCode house;
    int unit;
} X10LightDriverStruct;

//snip...
```

All **`LightDriver`** instances must start with the base **`struct`**

www.wingman-sw.com
james@wingman-sw.com

Initializing the vtable (X10LightDriver.c)

```
static void turnOn(LightDriver super)
{
    X10LightDriver self = (X10LightDriver)super;
    formatTurnOnMessage(self);
    sendMessage(self);
}
static void turnOff(LightDriver super)
{
    X10LightDriver self = (X10LightDriver)super;
    formatTurnOffMessage(self);
    sendMessage(self);
}
static LightDriverInterfaceStruct interface =
{
    turnOn,
    turnOff,
    destroy
};
```

Initializing a Driver (X10LightDriver.c)

```
static LightDriverInterfaceStruct interface =
{
    turnOn,
    turnOff,
    destroy
};

LightDriver X10LightDriver_Create(int id, X10_HouseCode house,
                                  int unit)
{
     X10LightDriver self =
                  calloc(1, sizeof(X10LightDriverStruct));
     self->base.vtable = &interface;
     self->base.type = "X10";
     /* snip */
     return (LightDriver)self;
}
```

Simple Version of LightDriver_TurnOn()

```
void LightDriver_TurnOn(LightDriver self)
{
    if (self)
        self->vtable->TurnOn(self);
}
```

The call is a bit hard to look at, so it's hidden in delegating function behind an API.

www.wingman-sw.com
james@wingman-sw.com

C99 struct Initialization

```
static LightDriverInterfaceStruct interface =
{
    .Destroy = destroy,
    .TurnOn = turnOn,
    .TurnOff = turnOff,
};
```


www.wingman-sw.com
james@wingman-sw.com

Really Safe Version of LightDriver_TurnOn()

```
void LightDriver_TurnOn(LightDriver self)
{
    if (self && self->vtable && self->vtable->TurnOn)
        self->vtable->TurnOn(self);
}
```

Combining safe function dispatch with C99 struct initialization means that not all drivers have to support all interface functions.

www.wingman-sw.com
james@wingman-sw.com

LightController uses the LightDriver

```
void LightController_TurnOn(int id)
{
    if (isIdInBounds(id))
        LightDriver_TurnOn(lightDrivers[id]);
}
```

- The switch case statement is gone! As well as all its duplicates.
- Code reads top to bottom
- Special cases are isolated

Creating a Specific LightDriver Using it Generically

Create parameters are customized for the specific driver type. It returns the abstract type

```
LightDriver lightDriver;

lightDriver = X10LightDriver_Create(3, X10_A, 12);

LightDriver_TurnOn(lightDriver);
LightDriver_TurnOff(lightDriver);
LightDriver_Destroy(lightDriver);
```

Clients of the driver have no dependency on the concrete type

www.wingman-sw.com
james@wingman-sw.com

Dynamic Interfaces are Very Convenient for Inserting Test Doubles

```
// From LightDriverSpy.c

static LightDriverInterfaceStruct interface =
{
    .Destroy = LightDriverSpy_Destroy
    .TurnOn = LightDriverSpy_TurnOn,
    .TurnOff = LightDriverSpy_TurnOff,
};
```

Production Code is Unaware of the Indirection

```
TEST(LightDriverSpy, On)
{
  LightDriver lightDriverSpy = LightDriverSpy_Create(1);
  LightDriver_TurnOn(lightDriverSpy);
  LONGS_EQUAL(LIGHT_ON,  LightDriverSpy_GetState(1));
}
```

- LightDriver_TurnOn() indirectly calls the spy

Function Pointer APIs

- One module operates on another module through a function pointer table.
- Allows a whole set of functions to be swapped with a single pointer change.
- Good for large systems with component variations
- Also good for updating or patching single components

- Choose the simplest approach. Keep code readable.

Testability

- The function pointer based APIs allow test doubles to be inserted at runtime.
- Legacy code that has function pointer APIs can be easier to get into a test harness.
  - Fill the function pointers with NULLs
  - Run until crash to see which functions are accessed
  - Add stubs as runtime dependencies are discovered
- All these design models employ programming to interfaces and encapsulation

C++

- The use of function pointers and function pointer tables does lead some to look at C++.
- C++ has this capability built in.
- Maybe you should look at C++.

- Why are you still using C?
  - http://renaissancesoftware.net/papers/14-papers/51-stillusingc.html

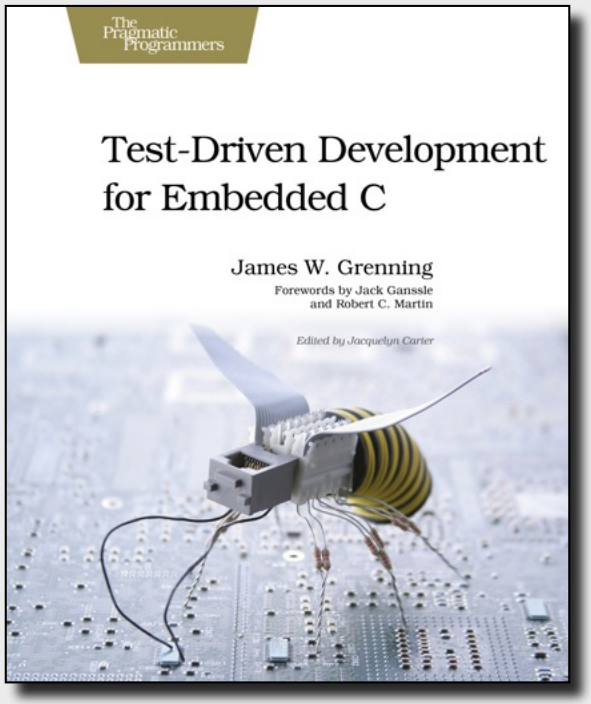
WINGMAN SOFTWARE

Ask me for the pragprog.com discount code

Talk to me on Twitter
@jwgrenning

Connect with me on linkedin.com
http://www.linkedin.com/in/jwgrenning
Remind me how we met.

http://www.wingman-sw.com
http://blog.wingman-sw.com
http:// www.jamesgrenning.com
unpappd.com jwgrenning



www.wingman-sw.com
james@wingman-sw.com