Agile Embedded Software Development
a Renaissance

Embedded Systems Conference
Boston October 2008 -- ESC 206
James W. Grenning
Renaissance Software Consulting Company
www.RenaissanceSoftware.net

What are your burning software development issues?
Software Development is Easy!

• Just like this *Black Diamond*

![Black Diamond Image](image)

We never have any problems like

• Late Delivery
• Poor Quality
• Burnout
• Missed Customer Expectations

Software Development is Easy!
Like this Cliff!

We Make Our Problems

Vague Requirements

↓
Unrealistic Plan

↓
Unplanned Activities

↑
Unpredictable Activities

→ Late Project

→ Missed Customer Expectations

↑
Poor Quality
Positive Feedback
Unstable System

- Vague Requirements
- Unrealistic Plan
- Unplanned Activities
- Unpredictable Activities

→ Misunderstood Requirements
→ Late Project
→ Schedule Pressure
→ Long Hours
→ Bugs
→ Missed Customer Expectations
→ Poor Quality

Can Projects be Managed Better?

Figure out everything, then do what you figured

From Winston Royce's original paper on Waterfall
Waterfall -- The Experience

March May July September November

Requirements Design Code Test

Time

Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix Test and Fix

Figure it All Out, Then Do It

Drive:

1,121 mi (about 17 hours 43 mins)
Why Consider Agile?

• Manage with data
• Improve Predictability
• Improve Quality
• Improve Productivity
Change Becomes Visible

New features added and nothing removed

But Change is Hard!
What is Agile?

• Agile software development is a conceptual framework for undertaking software engineering projects.
  -- wikipedia

• a.k.a. Extreme Programming, Scrum, Feature Driven Development, DSDM, Crystal Clear, Agile Unified Process

www.agilemanifesto.org

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.
Individuals and Interactions
over Processes and Tools

Working Software
over Comprehensive Documentation

- Each team has different needs
- Less formal documentation might work.
- Prefer executable Documentation
Customer Collaboration over Contract Negotiation

Responding to Change over following a plan

Drive:

1,121 mi (about 17 hours 43 mins)
Agile Practices Support the Iterative Model

- Iterative and Incremental Development
- Concurrent Engineering
- Teamwork
- Continuous Planning/Tracking
- Fine-Grain Scope Control
- Evolutionary Design
- Automated and Continuous Testing

Feedback

Renaissance Software Consulting

Iterative and Incremental Development

Projects end, products don’t (hopefully)

Requirements analysis is never done
Design is never done
Iterative and Incremental Development

Parallel instead of Serial

Working software delivered each iteration.

Track Projects Based on Functionality Built and Tested
Teamwork

- Shared workspace (lab)
- Collaborative development
  - Pair programming
- Accessible customer
  - Usually an internal customer
- Shared code ownership
- Continuous Integration
Concurrent Engineering

Requirements Engineering

Software Engineering

Hardware Engineering

Software Hardware Sequential Engineering

Requirements
Fine-Grain Scope Control

• Story
  – Something the system must do
  – Like a use case, part of a use case, or scenario

• Stories
  – Provide value
  – Demonstrate progress
  – Reduce risk
  – Can be estimate
  – Can be tested
  – Many fit into an iteration

Stories and Acceptance Tests

• Stories lack detail
• Details are provided in automated acceptance tests
• The test are like executable use cases
• Test either pass or fail
Getting Technical

Evolutionary Design

- All designs evolve
- Change is a fact of life
- Team works towards an Architectural Vision
- Details are worked out JIT in code
- Good designers needed hands-on
- Designs evolve using Refactoring techniques
- Bigger teams need more formality
Automated Testing

• Key to evolutionary design is keeping the cost of retest low
• 25% of defects are introduced while changing existing code
• Automated tests keep that cost low
• Test are run with every change

Where is the Time Going?

• 50% to Debug is commonly claimed
• 25% of all defects are introduced while changing and fixing code [R.B Grady, Software Process Improvement]

• One messed up project…
  – 5 months in requirements
  – 3 months development
  – 6 one month test and fix cycles
The Physics of Debug Later Programming (DLP)

- As $T_d$ increases, $T_{\text{find}}$ increases dramatically
- $T_{\text{fix}}$ is usually short, but can increase with $T_d$
The Physics of Test Driven Development

- When \( T_d \) is short \( T_{\text{find}} \) is short
- It is so short that it is not even considered a bug

Example Automated Test - 
\texttt{sprintf()}

```c
TEST(sprintf, formatString)
{
    char buffer[20];
    memset(buffer, 0xaa, sizeof(buffer));
    LONGS_EQUAL(12, sprintf(buffer, "%s\n", "Hello World"));
    STRCMP_EQUAL("Hello World\n", buffer);
    BYTES_EQUAL(0xaa, buffer[13]);
}

TEST(sprintf, formatInt)
{
    char buffer[20];
    memset(buffer, 0xaa, sizeof(buffer));
    LONGS_EQUAL(12, sprintf(buffer, "%i\n", 10));
    STRCMP_EQUAL("i=10\n", buffer);
    BYTES_EQUAL(0xaa, buffer[7]);
}
```
Program to Interfaces

- Separate interface and implementation as separate entities.
- This design has good separation of responsibilities.

Automated Test Example - Scheduler

```
TEST(LightScheduler, ScheduleWeekdayItsSunday)
{
    LightScheduler_ScheduleTurnOn(3, WEEKDAY, 1200);
    FakeTimeService_SetDay(SUNDAY);
    FakeTimeService_SetMinute(1200);
    FakeTimeService_MinuteIsUp();
    LONGS_EQUAL(-1, FakeLightController_getLastId());
    LONGS_EQUAL(-1, FakeLightController_getLastState());
}

TEST(LightScheduler, ScheduleWeekdayItsMonday)
{
    LightScheduler_ScheduleTurnOn(3, WEEKDAY, 1200);
    FakeTimeService_SetDay(MONDAY);
    FakeTimeService_SetMinute(1200);
    FakeTimeService_MinuteIsUp();
    LONGS_EQUAL(3, FakeLightController_getLastId());
    LONGS_EQUAL(1, FakeLightController_getLastState());
}
```
Specific Benefits for Embedded

- Reduce risk by verifying code independent of hardware
- Reduce long compile/link/load cycles
- Reduce debugging on target
- Isolates and models HW/SW interactions
- Improves design by isolating hardware dependencies
- Improves portability
- Develops a test safety net
More Embedded TDD Resources

• TDD in C Exercise
  – Please send me an email.
• www.renaissancesoftware.net/blog
  – TDD device driver example
  – Others
• Previous ESC presentations on my website
• yahoogroups.com/AgileEmbedded

Continuous Testing

• Testing starts on day one
• Tests provide the specification of what is to be developed
• QA/System Test moves upstream.
What’s Wrong With Waterfall?

Why Change? What's Wrong with Waterfall?

• Recommended Reading
  – Iterative and Incremental Development: A Brief History [LARMAN]
Waterfall Projects Fail at a High Rate

Jarzombek99 Study - Project Success/Failure

- 46% Never used or project failure
- 34% Other (including worked fine)
- 20% Extensive rework

What Happens when the Requirements are Committed Too Early?

- 45% Never used
- 16% Rarely used
- 13% Sometimes used
- 7% Often used
- 13% Always used
What’s Right about Iterative?
Why Iterative?

- A system's users seldom know exactly what they want and cannot articulate all they know
- … There are many details we can only discover once we are well into implementation
- … as humans we can only master only so much complexity
- … external forces lead to changes in requirements…

[LARMEN]

Iterative is Not new
Iterative is Not New

Agile is the name of the Renaissance that software development is experiencing.
Review Questions and Comments

• Agile Practices Support the Iterative Model
  – Iterative and Incremental Development
  – Concurrent Engineering
  – Teamwork
  – Continuous Planning/Tracking
  – Fine-Grain Scope Control
  – Evolutionary Design
  – Automated and Continuous Testing

Feedback