Real Time and Embedded Systems

by

Dr. Lesley Shannon

Email: lshannon@ensc.sfu.ca

Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc351

Adapted from guest lectures by Chris Simmons and the Extreme Programming Home Page

Simon Fraser University

Slide Set: 4

Date: October 4, 2011

Slide Set Overview

- How to tackle a BIG programming Problem
  - Software Engineering Considerations
- Extreme Programming
- How to draw threads/mutexes with CGN

How to tackle a ***big*** programming problem?

Software Engineering

- ***<u>HAVE A PLAN!!!</u>***
- Different software design methodologies:
  - Waterfall Method (older/traditional)
  - Agile

What is the Waterfall Method?

- W.W. Royce introduced a phased, linear software development methodology in 1970
  - He didn’t call it the “waterfall” method
  - Presented it as a flawed methodology
- Emphasizes:
  - Detailed project specs (Big “up-front” design)
  - ***Lots*** of Documentation on deliverables

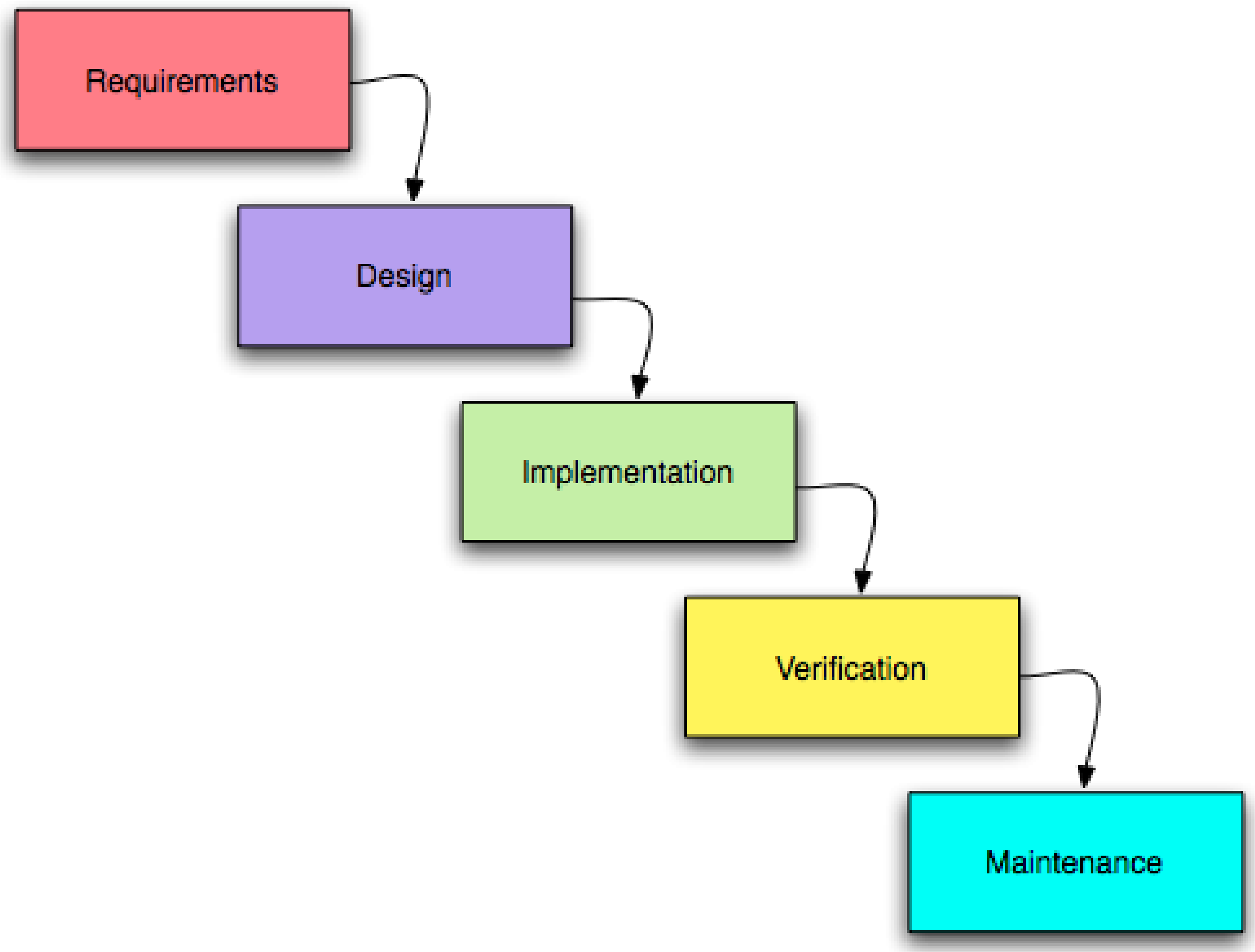
What is the Waterfall Method?

- Widely used in the "real" world
  - 44% of companies according to recent survey

  http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1318992,00.html

- Phases:
  1. Requirements Specification
  2. Design
  3. Construction
  4. Integration
  5. Testing and Debugging
  6. Installation
  7. Maintenance

The Waterfall Method



Benefits of the Waterfall Method

- More “up front” time is good for some apps:
  - e.g. safety critical, protocol specs
- Documentation is good
  - particularly for large safety critical apps
- Well defined deliverables
- Good for projects that don’t change
- Easier to understand long term requirements

Problems with the Waterfall Method

- Requirements aren’t always known up front:
  - Customers don’t know “exactly” yet
  - Market Changes
- Integration and testing done very late
  - Design Problems can be disastrous
- Customers don’t see it until you are done
  - Sounds good but customers “change their mind”
- Remember: described as a flawed method

Now Agile Software Development …

WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com scottadams@aol.com
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.

© Scott Adams, Inc./Dist. by UFS, Inc.

Agile “Manifesto”

- The Agile software development methodology “values”:
  - Individuals and interactions over process and tools
  - Working software over comprehensive documentation
  - Customer collaboration over negotiation
  - Responding to change over following a plan

Agile Software Development

- Emphasis on iterative development
  - 1-4 week blocks of time
  - Portions of the various Waterfall phases are combined into each iteration
- Each iteration ***should*** be a stable deliverable
  - Think latest software releases/patches
- No big upfront design
  - Design spec updated dynamically as needed
- And …

Agile Software Development

- Close contact with the customer
  - Or at least their “representative …
- THIS IS BOTH GOOD AND BAD IDEA

Agile Software Development

- Close contact with the customer
  - Or at least their “representative …

THIS IS BOTH GOOD AND BAD!!!

This leads us to Extreme Programming …

Extreme Programming

Why do we care about software engineering?

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."

- Author **Rick Cook**, *The Wizardry Compiled*

Extreme Programming

- A weird name for a good set of practices
- Created by Kent Beck (mid-90’s)
  - Checkout: *Extreme Programming Explained* (1999)
- An implementation of an Agile software development methodology
  - Takes Agile practices to their logical limit

Extreme Programming “Values”

- Communication (is “good noise”)
  - Know what’s going on with the team
- Simplicity
  - Do the simplest (not easiest!) thing that works
- Feedback
  - Retrospectives, unit tests, customer interaction

Extreme Programming “Values”

- Courage
  - Don’t be afraid to refactor/throw away code
- Respect
  - Good environment with reasonable hours

XP (practices)

- Fall into four main practices:
  1. Fine Scale Feedback: are we doing the right thing
  2. Shared Understanding: do we all agree on how to work
  3. Continuous Process: small improvements are better than nothing
  4. Programmer Welfare: aka work-life balance

XP practices: Fine Scale Feedback

- Includes:
  1. Pair Programming: Driver/Navigator model
  2. Planning “Game”:
     1. Release planning and
     2. Iteration Planning
  3. Test Driven Development

XP practices: Shared Understanding

- Includes:
  1. Coding Standards : K&R C, Code Conventions for Java, etc
  2. Collective code ownership:
     - ***<u>EVERYBODY is responsible for the code base</u>***
     - There is no such thing as “your code”
  3. Simple Design: Do the simplest thing possible that works (prevents “Big Upfront Design”)
  4. System metaphor: aka naming conventions

XP practices: Continuous process

- Includes:
  1. Continuous integration: much easier with revision control systems and automatic builds
  2. Design improvement: counterbalances “simplest first” with “refactor complex code”
  3. Small releases: prevent “going dark” encourage early feedback, not necessarily for customers

XP practices: Programmer Welfare

- Well rested employees are better employees, so overtime should be rare and spread out

I WORKED AROUND THE CLOCK AND FINISHED A PROJECT THAT WOULD NORMALLY REQUIRE TEN PROGRAMMERS.
UM... DID I JUST ESTABLISH A NEW BASELINE EXPECTATION THAT WILL TURN MY JOB INTO A TRAGIC DEATH MARCH?
IT'S TIME TO SET SOME STRETCH GOALS.
STUPID! STUPID! STUPID!
www.dilbert.com scottadams@aol.com
9-12-08 © 2008 Scott Adams, Inc./Dist. by UFS, Inc.

Extreme Programming

- Covers
  1. Planning
  2. Designing
  3. Coding
  4. Testing
  5. ***Managing***
- For the purpose of the project the first two steps are done for you
  – Only need to worry about coding and testing

Extreme Programming - Designing

1. Simplicity
2. Choose a system metaphor
3. No functionality is added early
   – Use customer feedback to focus/tune next release
4. Refactor whenever and wherever possible
   – Counterbalance to get simple things done first
   – Simplify code as you add new features making operations more complex

Extreme Programming - Coding

1. The customer is ***<u>always available</u>***
   – Involved in the whole process
2. Code must be written to agreed standards
   – K&R C, Code Conventions for Java, etc.
3. Code unit test first
   – Test Driven Development
4. All production code is pair programmed
5. Only one pair integrates code at a time
   – Driver/Navigator model

Extreme Programming - Coding

6. ***Integrate often***
   - Almost requires revision control and automatic builds (makefiles/etc)
7. Use collective code ownership
   - Everybody is responsible for the code base
     - Not just “your code"
8. Leave optimization till last
   - Do the simplest thing that works first and build on it

9. No overtime

I WORKED AROUND THE CLOCK AND FINISHED A PROJECT THAT WOULD NORMALLY REQUIRE TEN PROGRAMMERS.
UM... DID I JUST ESTABLISH A NEW BASELINE EXPECTATION THAT WILL TURN MY JOB INTO A TRAGIC DEATH MARCH?
IT'S TIME TO SET SOME STRETCH GOALS.
STUPID! STUPID! STUPID!
www.dilbert.com scottadams@aol.com
9-12-08 © 2008 Scott Adams, Inc./Dist. by UFS, Inc.

Extreme Programming - Testing

1. All code must have unit tests

2. ***<u>All code must pass all unit tests</u>*** before it can be released
   – This should prevent you from “going dark”
3. When a bug is found tests are created

4. Acceptance tests are run often and the score is published

Extreme Programming Reference

- Check out the web page:
  - www.extremeprogramming.org
- Planning to have a guest speaker (Chris Simmons) from industry (Sophos) come and talk about actual commercial uses of this technique
  - Attendance to their talk will be ***mandatory***
  - You will be responsible for the material he covers

Limitations of Extreme Programming

- It can break down in large groups
- Management needs to buy in to the idea
- Customer can be the single point of failure
- Teams must communicate
- Lack of documentation can clash with regulations
- Taken to the “extreme” can cause problems

XP for School: Least Useful Parts

- Planning game:
  - Release and iteration planning is often overkill
    - Exceptions: capstone projects/452/etc
- Since the requirements are fixed, don’t require close contact with the customer (me)
  - Remember there’s only one of me
- Design Improvement
  - You’ll likely never see coursework code again

XP for School: Medium Useful Parts

- Coding standards
  - Good for shared code development and coops
- Pair programming
  - Applicable for some courses and often a better learning experience
- Sustainable Pace
  - Lots of Classes, keep the work balanced

XP for School: Medium Useful Parts

- Collective Code Ownership
  - One person is ***NOT*** responsible for the project
- System Metaphor
  - Improves Code readability

XP for School: Very Useful Parts

- Test Driven Development
  - Using unit tests (when appropriate)
- Simple Design
  - KISS Rule: Do what you need to do to get the job done
  - Refactor and improve as you go
- System Metaphor
  - You'll be programming with another pair and you need to be able to read each others code

XP for School: Very Useful Parts

- Continuous integration
  - Revision Control becomes crucial
    - You’ll thank me
- Small Releases
  - Coding all day without compiling is BAD
  - If you can’t get it all done, at least you have something to show
  - Finish the important features first

Other Software Engineering Thoughts

- Learn an editor
  - It will save you time in the long run
    - Can be incorporated into IDE/Eclipse environments
- vim: http://www.vim.org
  - Extremely powerful and Concise
    - But the learning curve can be steep
  - Available on almost any unix/linux box (download for windows)

Other Software Engineering Thoughts

- Learn an editor (cont'd)
  - It will save you time in the long run
    - Can be incorporated into IDE/Eclipse environments
- Emacs: http://www.gnu.org/software/emacs/
  - Also powerful, but has a different input style
  - Also available on most boxes

Useful links for Unit Testing

- libtap:
  - http://jc.ngo.org.uk/trac-bin/trac.cgi/wiki/LibTap
  - Check: http://check.sourceforge.net
- Unit testing in almost any language:
  - http://www.testingfaqs.org/t-unit.html
- A list of papers on unit testing:
  - http://tinyurl.com/2ncqrf

Drawing threads sharing a semaphore

Drawing Threads Sharing a Semaphore

Drawing Threads Sharing a Semaphore

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Drawing Threads with Message Passing

Questions?

- Why use extreme programming techniques for commercial projects?

- How would you draw the CGN for the “Sleeping Barber” problem?