Software Tools & Methods
Class 2

Lecturer: Greg Bolcer, greg@bolcer.org

Summer Session 2009

ELH 110 9am-11:50am

Overview

- Homework?
- Last Class
  - Software Tools and methods
  - Ignorance in Software Requirements engineering
  - No Silver Bullet – essence of software
- Reverse Engineering and Refactoring
  - Reading assignment V. 14.3
- Up next
  - Software Process
  - Programming Practices
  - Code reading

Notes

- In-class portion (laboratory) meant to be an easy warm-up
  - Future labs will be more difficult
  - Just follow the instructions & you'll do fine
  - Doing labs at home?
- Please print out the cover page and bring it with you
  - TA will be using it for sign-offs
  - Don't forget the take home questions at the end

Software methods and tools

REVERSE ENGINEERING AND REFACTORING

Reverse Engineering and Refactoring

- **Reverse engineering** (RE) is the process of discovering the technological principles of a device, object or system through analysis of its structure, function and operation. It often involves taking something (e.g., a mechanical device, electronic component, or software program) apart and analyzing its workings in detail to be used in maintenance, or to try to make a new device or program that does the same thing without copying anything from the original. (wikipedia)
- Not just for stealing code, e.g de-compilation of Java
- “Software archeology”

Reverse Engineering

- Chikofsky (1990)
  - Identify the systems components and their interrelationships
  - Create representations of the system in another form or at a higher level of abstraction
- Typically done through inspection
- Useful for design recovery or redocumentation to help with future changes
- Lab today will show how tools can help with refactoring: name change of a package

Reverse Engineering

- What it's good for
  - Recovering legacy systems where no information about original environment or development process or procedures involved
  - Lost documentation
  - Product analysis
  - Improving quality
  - Security auditing and forensics
  - Circumvent copy restrictions
  - Emulation
  - Competitive technical evaluation
  - Curiousity
  - Learning from others mistakes

Reverse Engineering

- A completely reverse engineered copy is difficult to achieve
  - 100% functional equivalence difficult to achieve due to conformity issues
  - Resist urge to correct errors during process
  - Data equivalence may change due to typing issues, data conversions

Other Types of Re-engineering

- **Restructuring**: transformation of a system from one representation to another at the same level of abstraction
  - E.g. spaghetti code to structured code
- **Refactoring**: a white-box method that involves inspection and changes to code
- **Renovation**: Design changes are made with new work
- **Migration**: changing to run on a new platform
- Can also modernize code using a black-box refactoring
  - Code wrapping, e.g. having a text client run in a Web browser
  - Changing databases, XML document integration
  - Adding a GUI using a model-view-controller method

Redocumentation

- The derivation of a semantically equivalent description at the same level of abstraction
- Examples:
  - Transformation of a badly indented program into one having a neat layout
  - The construction of a set of flow charts for a given program

Refactoring

- Modern name for restructuring
- Popular practice for XP (extreme programming)
- Can use tools as simple as a debugger, automated call-diagram, package layout
- Against: reasons not to refactor
  - If it ain't broke, don't fix it
  - Operationally may re-trigger full testing
  - May break APIs or post-deployment integrations
  - Temptation to leverage newest technologies where not needed
  - Garbage in, garbage out
    - Refactoring an unstructured, unmodular mess through an automated refactoring tool will result in a structured, umodular mess

Reasons for Refactoring

- Improves quality, reduces inefficiencies
- Reduced complexity, dependencies
- Improved maintainability, extensibility
- Example problem areas
  - Class explosion—alternate classes that do similar things
  - Speculative generality—ambitious code for future features
  - Tightly couples classes and dependency cycles
  - Excessive switch or case statements
  - Very, very long methods
  - Paranoia of something bad happening

Refactoring Case Study

- Software + hardware system that parsed text messages at about 3,500 per second from network
  - Engineers built hardware RAID in software to guard against (unlikely) data loss failures
  - Design of system was synchronous and sequential to prevent any data getting out of order
  - Rule engine that looked for patterns in data would alert users within 100ths of a second when matches were found

Refactoring Case Study

- Able to review the fundamental requirements and developer assumptions
- Refactored storage, rule engine, throughput to preserve functional equivalence
  - Removed software RAID, used hardware
  - Parallelized operations where data wasn't coupled
  - Changed user notification to how users really work
- Went from 3,500 events to 60,000 events per second on like-for-like hardware
- Migrated to new platform and forward engineered scalability resulting in 300,000+ events per second

Software tools and methods

SOFTWARE PROCESS

Software Process

- **Software Process** is a set of activities whose goal is the development or evolution of software
  - Specific and enacted
- A **Software Process Model** is a simplified representation of the software process, presented from a specific perspective.
  - General and abstract
- Like the difference between class and object/instance

Software Process

- Typical stages of a software process include
  - Specified
  - Designed
  - Implemented

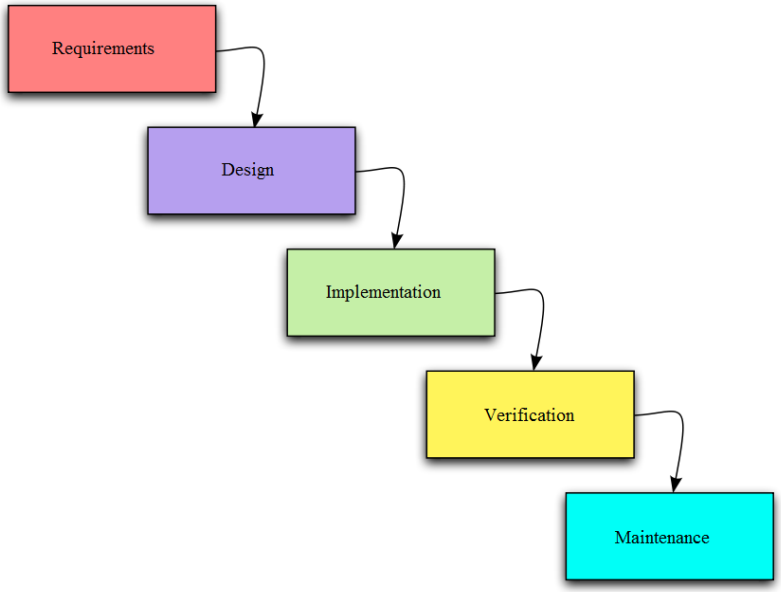
And once it's operational or deployed

  - Maintained

How Process Models Vary

- Phased or incremental
- Plan-based or iterative
- Continuous testing or late testing
- Feedback
- Risk management

Waterfall Model



- A sequential software development process in which progress is seen as flowing downward through phases
- Winston Royce (1970)
- Presented as a rigid and flawed development process

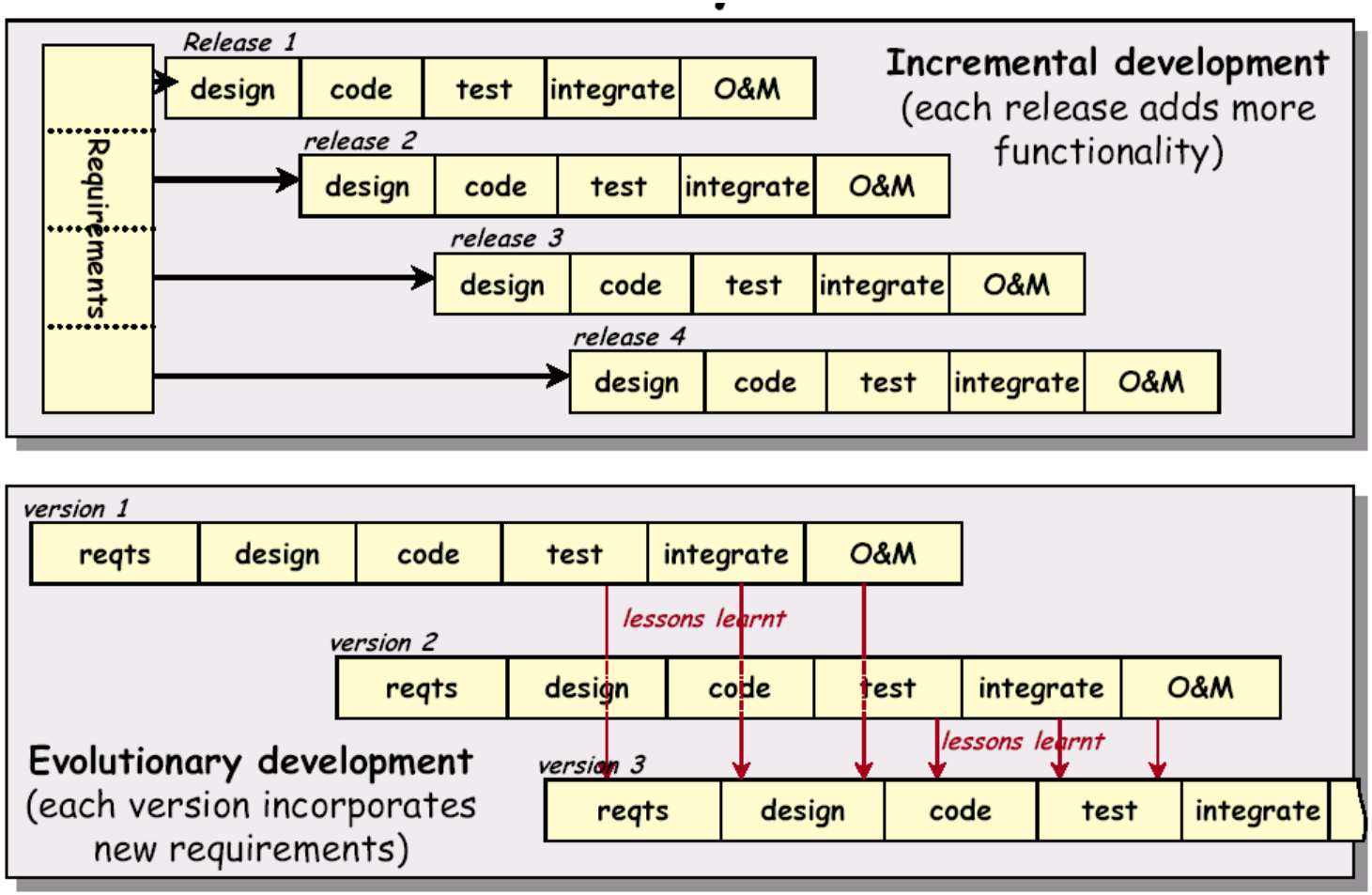
Waterfall Model

- Time spent earlier stages in the process have downstream benefits
  - Reduced time to fix bugs when caught earlier and sooner, up to 50x
  - Better design, emphasis on documentation and less on source code
  - Accuracy and thoroughness
- Good for stable software projects
- Used for projects with unchanging requirements
- Essentially true how software is designed and implemented
- Bad idea in practice for most non-trivial projects
  - Requirements do change
  - No project ever progressed this way
  - Difficult to lock down phases as complete
- Not all projects have complete information
  - what customers wants up front
  - Implementation details
- New models address shortcomings
  - Waterfall + feedback
  - Overlapping phases

Evolutionary Process Models

- Allows the software to evolve as need grows or requirements become better defined
- Each delivery becomes more complex with new features or functions
- Goal of Evolutionary models is extensibility
- Examples include
  - Prototyping model
  - Concurrent development
  - Spiral model

Evolutionary Model



Advantages

- Accommodates throw-away prototyping
- Allows for lessons from each version to be incorporated into the next

Disadvantages

- Hard to plan for versions beyond the first
- Lessons may be learned too late
- Process is not visible
- Systems are often poorly structured
- Special tools and techniques may be required

Prototyping Model

- Used when
  - Short amount of time for product
  - Needs revisions or updates after release
  - Requirements are fuzzy or not completely known
  - Developer is unsure of
    - The efficiency or scalability of an algorithm
    - The adaptability of the OS
    - User interface is not well defined

Prototyping Model

Advantages

- Delivers a working system early and cheaply
- Avoids building systems to bad requirements
- Engages customer early
- Fits top-down implementation and testing strategies

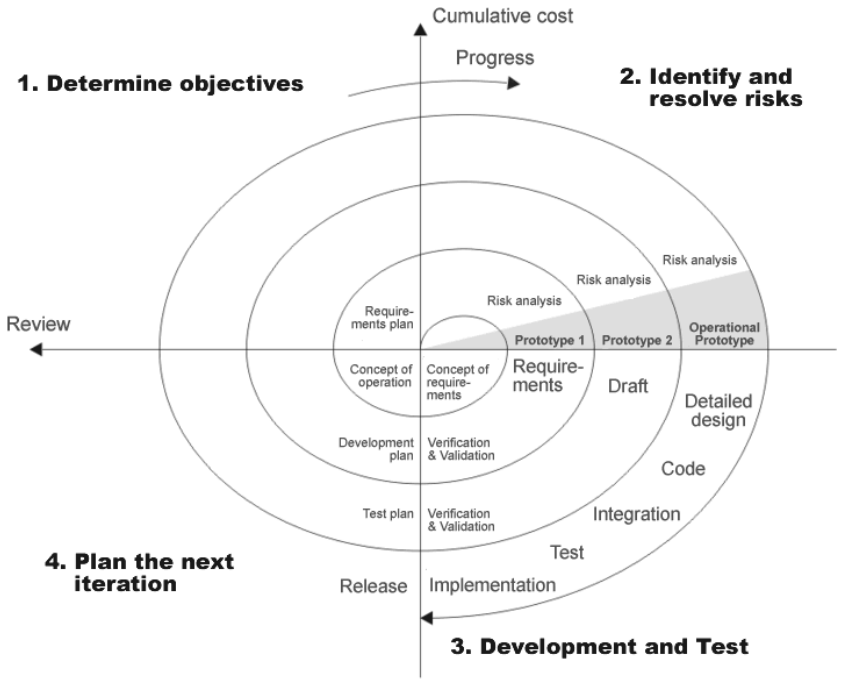
Disadvantages

- Users may become frustrated
- Software may be volatile
- System may be unreliable
- Mismatched expectations

Concurrent Development Model

- Developers write requirements, design, code, test, and integration tests all at the same time
- Any activities of a project may be in a particular state at one time
  - Under development
  - Awaiting changes
  - Under revisions
  - Under review, etc.
- Used to client/server applications
- Advantages: quick turnover of product
- Disadvantages: Interdependencies and miscommunication between different developers

Spiral Model



- Barry Boehm (1987)
- An evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model
- Before starting each phase, an attempt is made to reduce or resolve risks
  - If impossible to resolve all significant risks, project is rescoped/reworked
  - Good for large scale, complex software development

Spiral Model

Framework activities

- Customer communication – tasks to establish effective communication and feedback
- Planning – tasks to define resources, timelines, and project related information
- Risk Analysis – tasks required to assess the technical and managerial risks
- Engineering – tasks required to build one or more representations of the application
- Construction & Release – tasks required to construct, test, and support the software (docs, training)
- Customer evaluation – tasks required to obtain periodic customer feedback

Spiral Model

Advantages

- Emphasize risk which is most often ignored
- Risk reduction means less failed projects
- Check points for project getting off of track
- Constant customer involvement and validation

Disadvantages

- Full analysis requires training, skill, considerable expense
  - e.g. New business objectives
  - Not always clear how to analyze risk
- Best suited for only very large projects run by very large companies,
  - e.g. aerospace, military, government

Evolutionary Processes Summary

- Almost all modern software development is characterized by
  - Continual change
  - Tight deadlines
  - Need for user/customer satisfaction
- Evolutionary processes designed to meet exactly these issues

Whiteboard Exercise

- What is the difference between an iterative and incremental software development process model?

Whiteboard Exercise

- Iterative versus incremental
  - Both are cyclical models developed from limitations of the waterfall model
  - Incremental is a staging strategy where parts of the system are developed at different times or rates and are integrated as completed
    - Versus a “Big Bang” integration all at once in a single stage
  - Iterative is a rework strategy where certain time is set aside to revise and improve parts of the system
  - Can use both in the same process, e.g. Rational Unified Process, eXtreme Programming, Agile

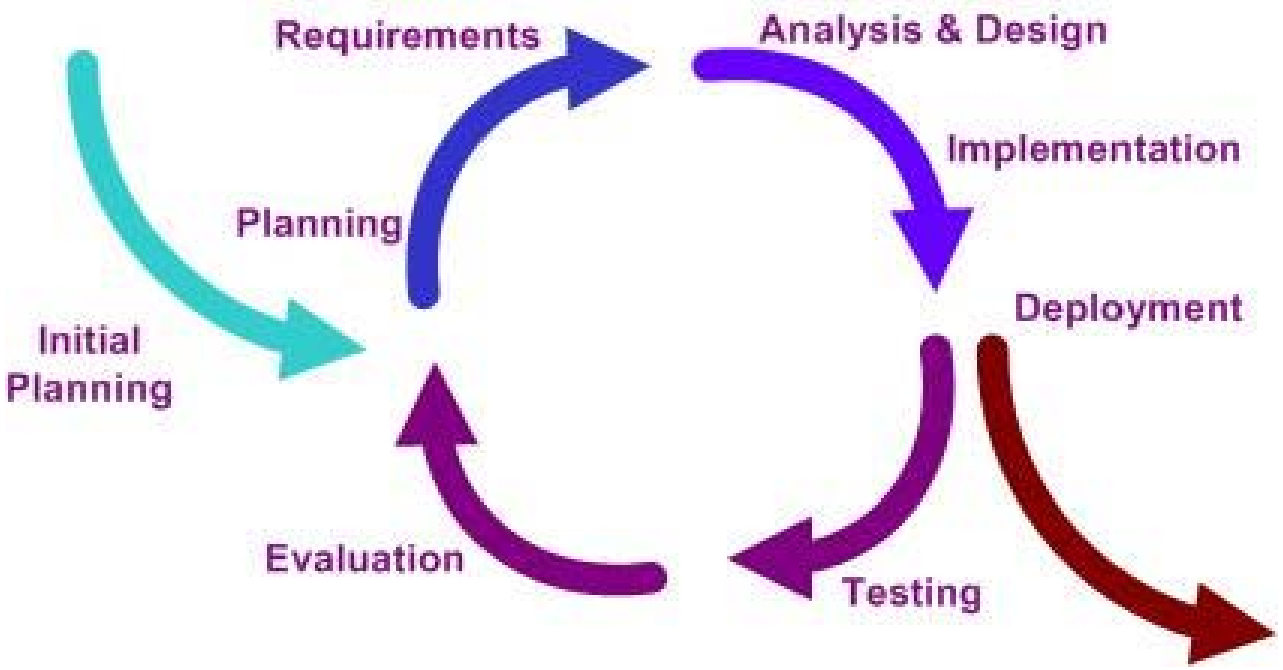
Agile Methods

- Currently, very popular in industry
- Agile means being able to move quickly
  - Mentally quick and resourceful
- Develop software iteratively and incrementally
- Manage risk by managing scope
  - Strong customer focus
- Continuous feedback
  - Between developers, managers, and customers

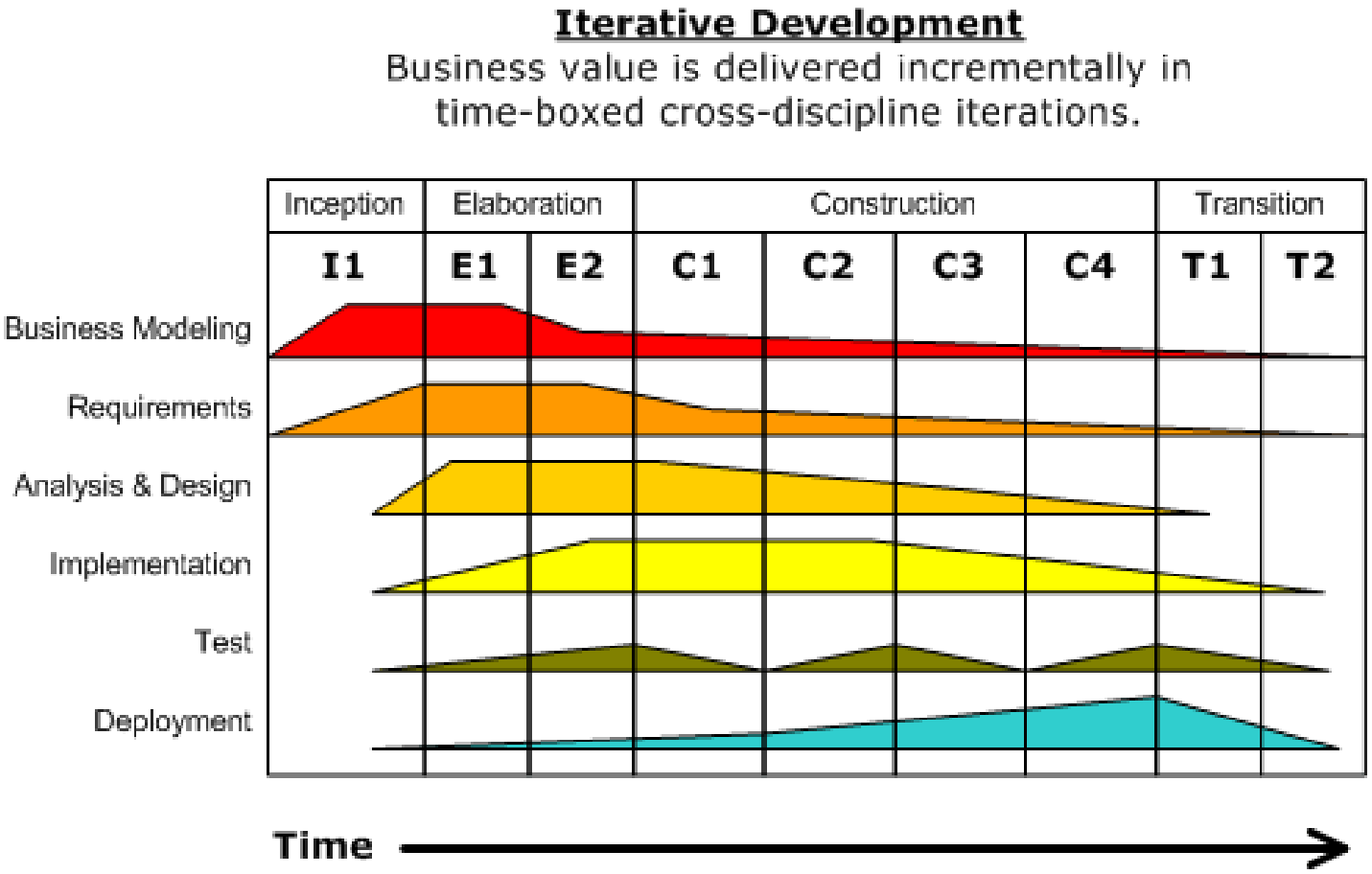
- Can not plan for all possible changes, instead embrace change

Agile Philosophy

Iterative and Incremental Models



Iterative Effort by Phases

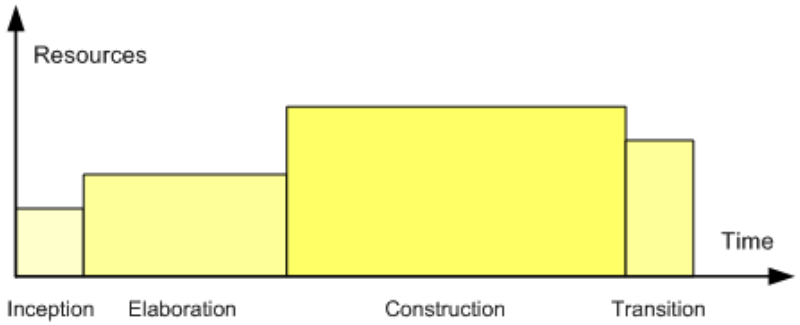


Rational Unified Process

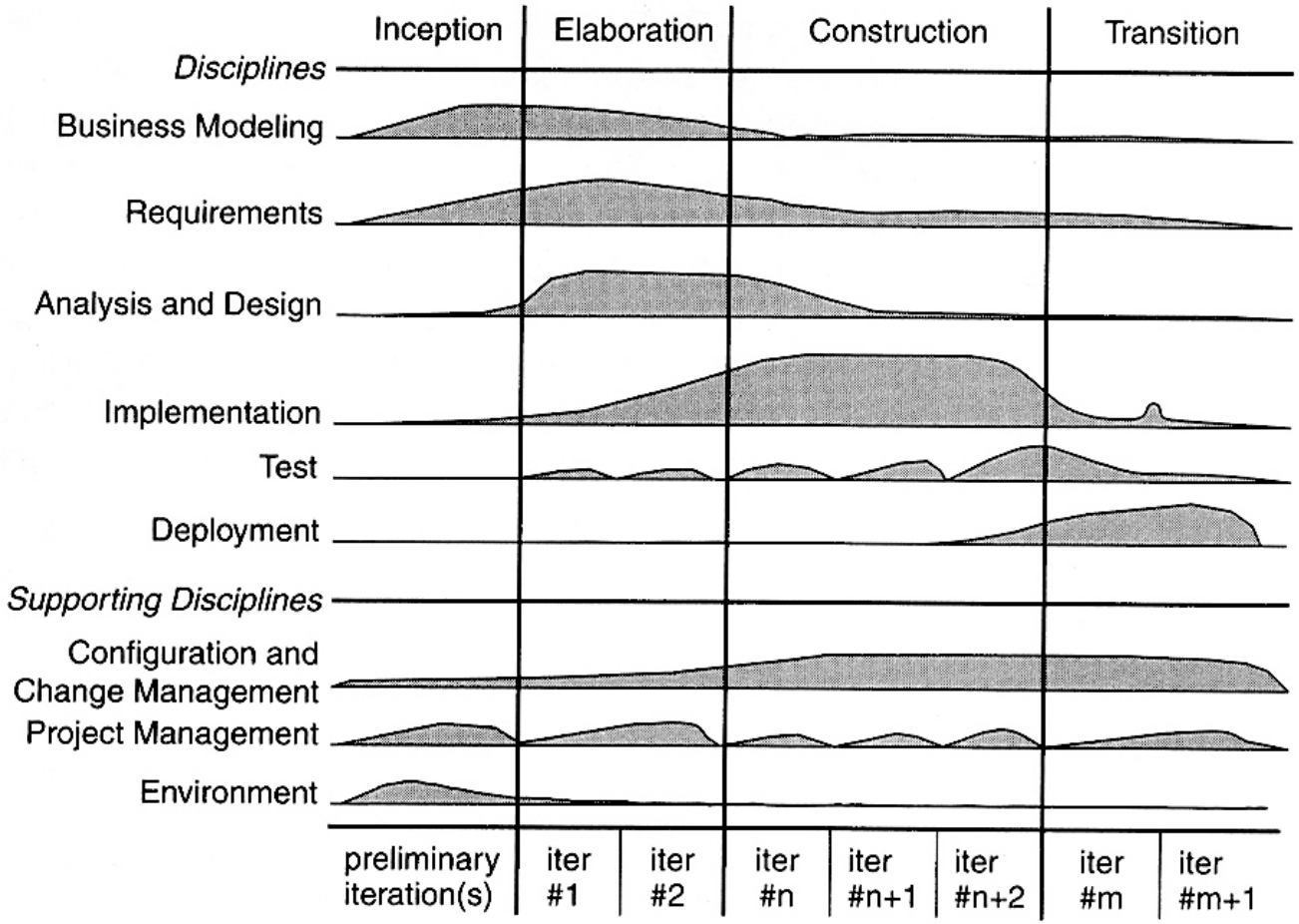
- Rational is a software development or design process that is:
  - Use case-driven
  - Architecture-centric
  - Iterative and incremental
  - Risk focused
- Framework stages
  - Inception
  - Elaboration
  - Construction
  - Transition

Rational Unified Process

- Phases based on what question you are trying to answer:
  - **Inception** - do you and the Customer have a shared understanding of the system?
  - **Elaboration** - do you have an architecture to be able to build the system?
  - **Construction** - are you developing product?
  - **Transition** - are you trying to get the Customer to take ownership of the system?



Rational Unified Process



eXtreme Programming (XP)

- Four Values
  - Communication, simplicity, feedback, and courage
- The Principles
  - Concrete applications of the principles
    - Rapid feedback; assume simplicity; incremental change; embracing change; quality work
- Four Basic Activities
  - Coding, testing, listening, and designing
- Twelve Practices

eXtreme Programming

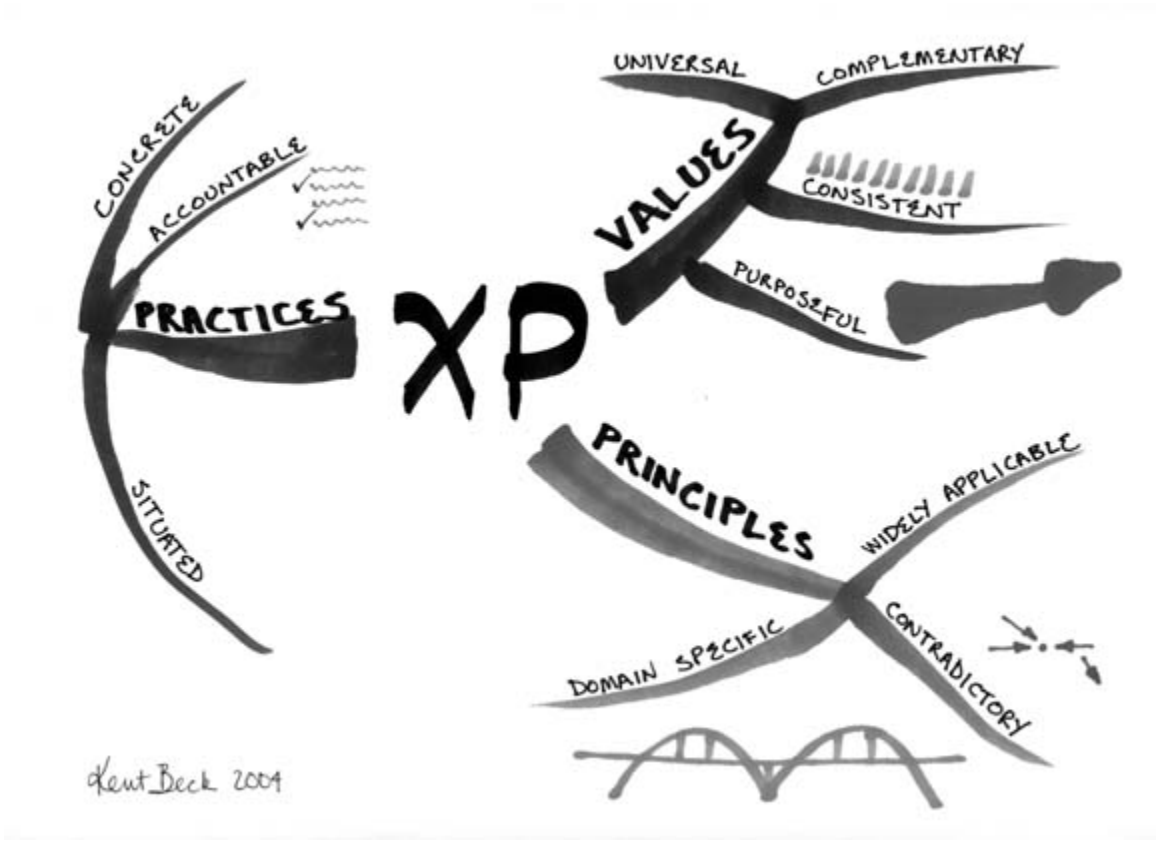
Look familiar?

eXtreme Programming

Not “Cowboy Coding”

Not “Programmer centric”
Or
“Ignore the customer”

XP is Purposeful



eXtreme Programming

- Code → Test → Listen → Design
- “extreme” comes from, if prototyping, incremental, and spiral models are good—more must be better
- Includes customer approved stories
  - Better quality and responsiveness to changing requirements
  - Test driven development—does it do what it’s supposed to?
- Frequent releases, Short timeboxes
  - Checkboxes where confirm doing the right thing
- Flat management structure
- Preference for simplicity of design and code
- Drawbacks
  - Unstable requirements
  - No documented compromises of decisions
  - Lack of overall spec design or documents

Scrum Development Process

Roles

Product Owner: Set priorities

ScrumMaster: Manage process, remove blocks

Team: Develop product

Stakeholders: observe & advise

Key Artifacts

Product Backlog
- List of requirements & issues
- Owned by Product Owner
- Anybody can add to it
- Only Product Owner prioritizes

Sprint Goal
- One-sentence summary
- Declared by Product Owner
- Accepted by team

Sprint Backlog
- List of tasks
- Owned by team
- Only team modifies it

Blocks List
- List of blocks & unmade decisions
- Owned by ScrumMaster
- Updated daily

Increment
- Version of the product
- Shippable functionality (tested, documented, etc.)

Key Meetings

Sprint Planning Meeting
- Hosted by ScrumMaster; ½-1 day
- In: Product Backlog, existing product, business & technology conditions

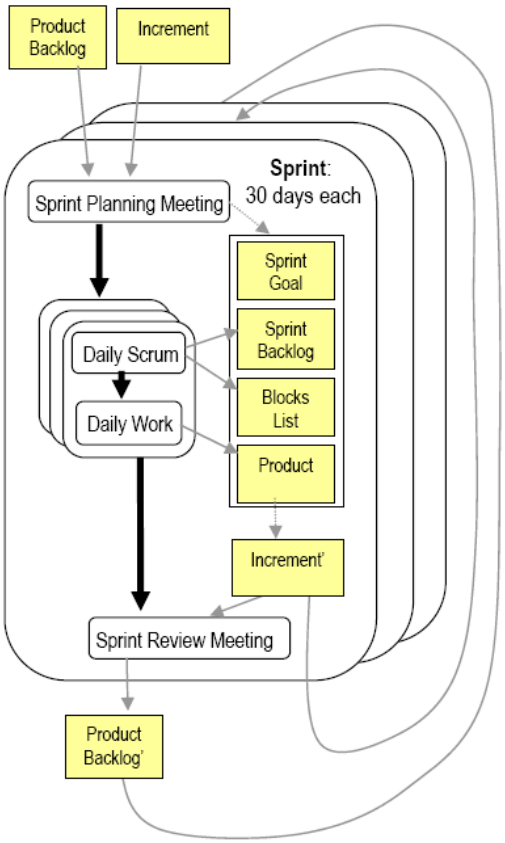
1. Select highest priority items in Product Backlog; declare Sprint Goal
2. Team turns selected items into Sprint Backlog

- Out:: Sprint Goal, Sprint Backlog

Daily Scrum
- Hosted by ScrumMaster
- Attended by all, but Stakeholders don't speak
- Same time every day
- Answer: 1) What did you do yesterday? 2) What will you do today? 3) What's in your way?
- Team updates Sprint Backlog; ScrumMaster updates Blocks List

Sprint Review Meeting
- Hosted by ScrumMaster
- Attended by all
- Informal, 4-hour, informational
- Team demos Increment
- All discuss
- Hold retrospective
- Announce next Sprint Planning Meeting

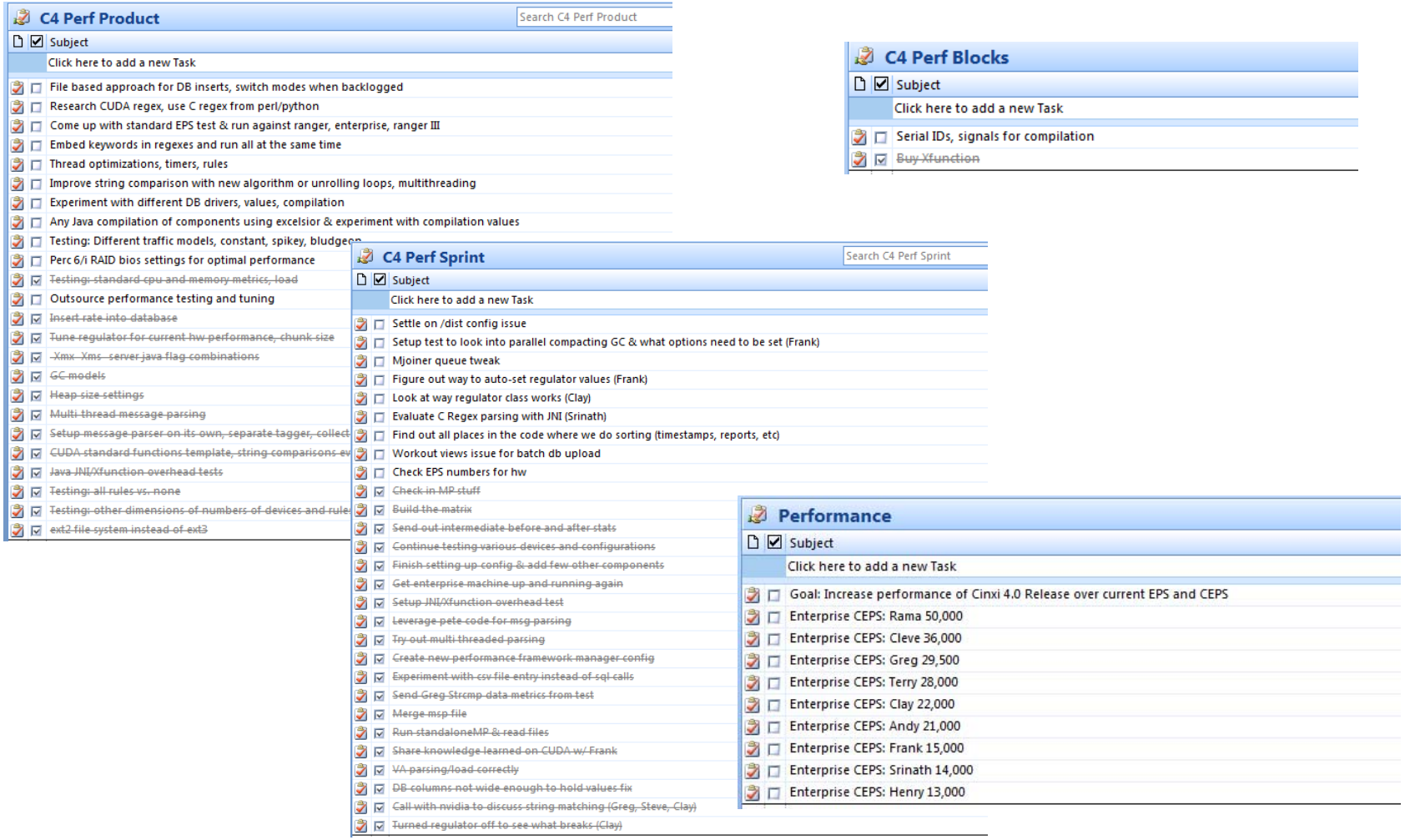
Development Process



Tricks to a successful Sprint Project

- Have a formal kickoff with a fixed time, eg. 10 – 30 days for first increment
- Use a tools for each of the documents
- Be the note-taker yourself
- Listen carefully to what everyone says
  - Pick out and write down the details no matter how small
  - It's okay to have some participants not do anything due to other projects or priorities on some days
  - Let everyone have their say
- Pass out paperwork and cross out tasks, blocks, and goals in the meeting
  - Include crossed out tasks in handouts

Scrum Case Study



C4 Perf Product

Search C4 Perf Product

Subject

Click here to add a new Task

- File based approach for DB inserts, switch modes when backlogged
- Research CUDA regex, use C regex from perl/python
- Come up with standard EPS test & run against ranger, enterprise, ranger III
- Embed keywords in regexes and run all at the same time
- Thread optimizations, timers, rules
- Improve string comparison with new algorithm or unrolling loops, multithreading
- Experiment with different DB drivers, values, compilation
- Any Java compilation of components using excelsior & experiment with compilation values
- Testing: Different traffic models, constant, spikey, bludgeon
- Perc 6/i RAID bios settings for optimal performance
- ~~Testing: standard cpu and memory metrics, load~~
- Outsource performance testing and tuning
- ~~Insert rate into database~~
- ~~Tune regulator for current hw performance, chunk size~~
- ~~-Xmx -Xms -server java flag combinations~~
- ~~GC models~~
- ~~Heap size settings~~
- ~~Multi thread message parsing~~
- ~~Setup message parser on its own, separate tagger, collect~~
- ~~CUDA standard functions template, string comparisons ev~~
- ~~Java JNI/Xfunction overhead tests~~
- ~~Testing: all rules vs. none~~
- ~~Testing: other dimensions of numbers of devices and rules~~
- ~~ext2 file system instead of ext3~~

C4 Perf Blocks

Subject

Click here to add a new Task

- Serial IDs, signals for compilation
- ~~Buy Xfunction~~

C4 Perf Sprint

Search C4 Perf Sprint

Subject

Click here to add a new Task

- Settle on /dist config issue
- Setup test to look into parallel compacting GC & what options need to be set (Frank)
- Mjoiner queue tweak
- Figure out way to auto-set regulator values (Frank)
- Look at way regulator class works (Clay)
- Evaluate C Regex parsing with JNI (Srinath)
- Find out all places in the code where we do sorting (timestamps, reports, etc)
- Workout views issue for batch db upload
- Check EPS numbers for hw
- ~~Check in MP stuff~~
- ~~Build the matrix~~
- ~~Send out intermediate before and after stats~~
- ~~Continue testing various devices and configurations~~
- ~~Finish setting up config & add few other components~~
- ~~Get enterprise machine up and running again~~
- ~~Setup JNI/Xfunction overhead test~~
- ~~Leverage pete code for msg parsing~~
- ~~Try out multi threaded parsing~~
- ~~Create new performance framework manager config~~
- ~~Experiment with csv file entry instead of sql calls~~
- ~~Send Greg Strcmp data metrics from test~~
- ~~Merge msp file~~
- ~~Run standaloneMP & read files~~
- ~~Share knowledge learned on CUDA w/ Frank~~
- ~~VA parsing/load correctly~~
- ~~DB columns not wide enough to hold values fix~~
- ~~Call with nvidia to discuss string matching (Greg, Steve, Clay)~~
- ~~Turned regulator off to see what breaks (Clay)~~

Performance

Subject

Click here to add a new Task

- Goal: Increase performance of Cinxi 4.0 Release over current EPS and CEPS
- Enterprise CEPS: Rama 50,000
- Enterprise CEPS: Cleve 36,000
- Enterprise CEPS: Greg 29,500
- Enterprise CEPS: Terry 28,000
- Enterprise CEPS: Clay 22,000
- Enterprise CEPS: Andy 21,000
- Enterprise CEPS: Frank 15,000
- Enterprise CEPS: Srinath 14,000
- Enterprise CEPS: Henry 13,000

Scrum Case Study

- Legacy project and staff paralyzed by over-specification, infighting and dependencies
- Continued regular development on overall product
- Assembled series of goals, one issue at a time: performance, platform change, live updates
- Ran a sprint project for each goal with different teams
- Created a competitive atmosphere
- Keep to a standard work week < 50 hours, no weekends

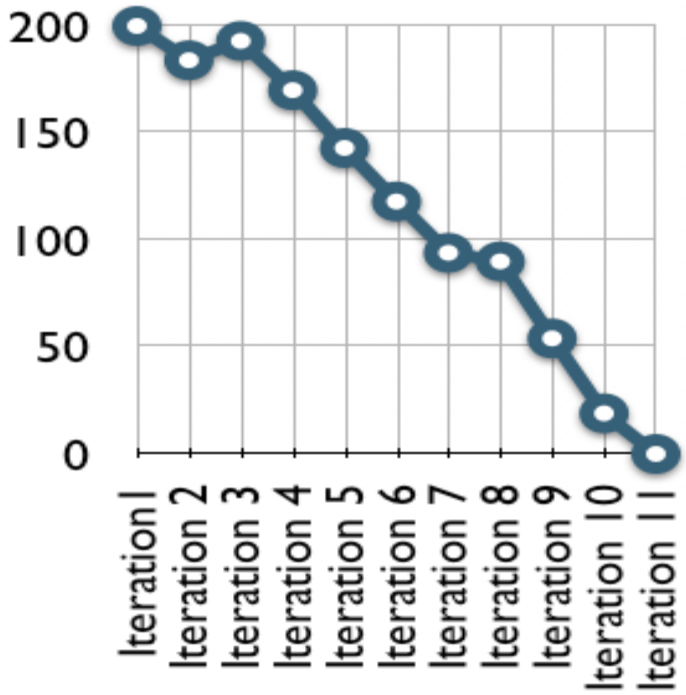
Scrum Pushbacks

- Programmer pushback
  - "This is stupid"
    - Nothing more enjoyable than seeing developers transform in the middle of a project and get excited about what they are doing and having a sense of ownership and accomplishment
  - Resent visibility into work habits
    - Very uncomfortable for some staff, to the point of resigning, but need to factor that in when scoping the project
  - "My manager will go to the meeting for me"
    - Need the people actually doing the work—flat organization
  - "These are just like my daily meetings"
    - Productive vs Unproductive, the key is the structure and supporting documents and tasks

Scrum Pushbacks

- Product and Business pushback
  - "How do you keep track of how far along the project is?"
    - It's a 30 day project, when we're 15 days in, it'll be halfway done
  - "How do you know if they'll meet their deadlines?"
    - On day 30, we'll be able to see what goals have been completed and which haven't
  - "How do we grant them an extension?"
    - No extensions—it's pass or fail
  - "What if they fail? Won't that hurt the business?"
    - The scope of the project is such that if it succeeds, the business succeeds, if it fails, we factored in the risk and we try something else
  - "I want to be in the meetings."
    - No. Pigs and Chickens. My meeting, my scope, my process model.

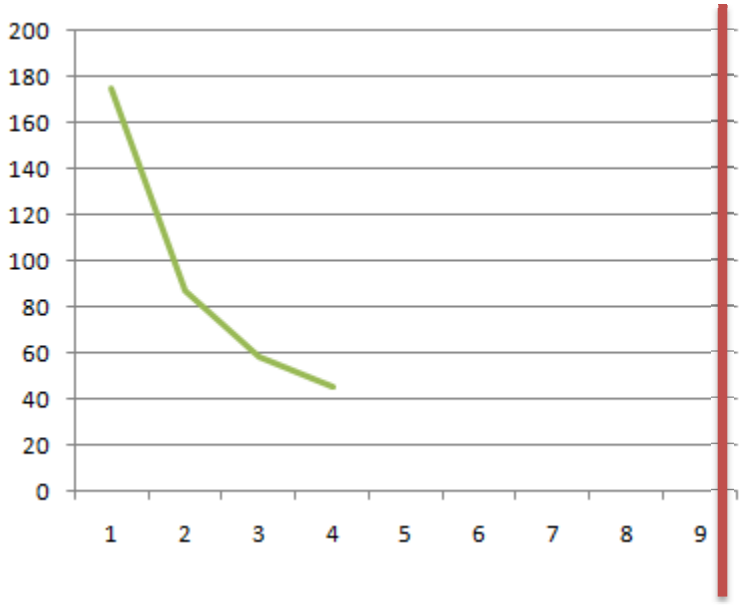
Scrum Burndown



- Useful to also predict
  - Stability of software
  - Resource planning
  - Estimated completion date
  - Problem areas

Whiteboard Exercise

Sprint Backlog items vs. Week



Deadline, end of week 9

What week will the sprint backlog be completed?

When will the Sprint project end?

Software methods and tools

PROGRAMMING PRACTICES

Programming Practices

- Best practices are “good ideas” learned from experience
- Starting point for gaining expertise
  - Basic habits need to be ingrained, so you can focus on essential difficulties
  - Since you’re going to form habits, make them good ones instead of bad ones
    - Tools that you can always reach for, and know where they are instinctively

Code Convention

- A set of guidelines and rules for
  - Formatting, labeling, and structuring the code
  - Commenting code
  - Naming convention for files, variables, methods
- Java Code Convention
  - A coding standard that Sun recommend
  - Includes some programming practices
  - http://java.sun.com/docs/codeconv/

Formatting Rules

- Method declarations should be grouped by functionality instead of scope or accessibility.
- Four spaces should be used as the unit of indentation
- Avoid lines longer than 80 characters
- One declaration per line is recommended.
- Do not put different types declaration on the same line, e.g. int foo, fooarray[];

Comments

- Comments should be used to give an overview of code and provide additional information that is not readily available in the code itself
  - Don’t Repeat Yourself
- The comment block before a method should include
    - Why it exists
    - Pre-Conditions and Post-Conditions
    - Exceptions – What can go wrongs and what would happen in that case.

Naming Convention

- Package
  - The prefix of a unique package name is always written in all-lowercase
  - Usually the domain name reversed
- Class Names
  - Should be nouns in mixed case (first letter of each internal word capitalized)
  - Try to keep your class names simple and descriptive, avoid acronyms
- Method Names
  - Should be verbs, in mixed case with the first letter
- Class Constants
  - The should be all uppercase with words separated by underscores

Why Do We Need Code Conventions?

- Improve the readability of the software
  - Takes less time to understand or remember (if it's your own code) what the code does
  - 80% of the lifetime cost of a piece of software goes to maintenance.

What is Code Reading?

- Process of understanding computer source code
  - Code may or may not have documentation
  - Code may or may not compile/run
  - You may have additional help, e.g. people
- Goals
  - What the program is supposed to do
  - How it does them
  - Why it does them that way.

It not just about the code

- Need to understand other documents— e.g. software specifications or designs document if available
- Need to understand program existing structure and decide decision
- Can be done interactively or dynamically, e.g. with debugger or using profiler trace

Homework 2

- Lab: using Eclipse for Java projects
- Take home

1. **Discuss similarities and differences between Prototyping and Incremental Development process. (20 points)**
2. **Compare and contrast the Waterfall model and iterative software process models. (30 points)**
3. **Discuss the benefits of using a coding convention during software development (10 points)**