CS3505/5020
Software Practice II

C# Style Guides
Software processes
Agile methods
C# Programming Style


- See: http://www.irritatedvowel.com/Programming/Standards.aspx as it provides examples and reasons why

- Capitalization
  - Pascal – BouncingSprite
  - Camel – bouncingSprite
  - UpperCase – BOUNCINGSPRITE

- Hungarian Notation (bad)
  - m_size – indicates size is a member variable
  - i_size – indicates size is an integer
C# Programming Style - 2

● Rule 1:
  – Do use Pascal casing for all *public* member, type, and namespace names: `RelativeCenter`

● Rule 2:
  – Do use camel casing for parameter names: `imageFile`

● Rule 3:
  – Do capitalize only the first character of acronyms with three or more characters for public members, types, and namespaces: `XmlText`
C# Programming Style - 3

● Rule 4:
  – Do not capitalize any of the characters of any acronyms, whatever their length, at the beginning of a camel-cased identifier: `xmlText`

● Rule 5:
  – Do name classes, interfaces, properties, and value types with nouns, noun phrases, or occasionally adjective phrases, using Pascal casing: `RotatingSprite`

● Rule 6:
  – Do name methods with verb or question (for bool) phrases: `ComputeDepth, isHome`
C# Programming Style - 4

- **Rule 7:**
  - Consider ending the name of derived classes with the name of the base class: `CheapSprite`

- **Rule 8:**
  - Do prefix interface names with the letter `I` to indicate that the type is an interface: `IUpdate`

- **Rule 9:**
  - Constants should be the same rule as above:
    
    ```c#
    public const int BoardLeftLocation = 100;
    ```
C# Programming Style - 5

● Anti-Rule 1:
  – NO SCREAMING DECLARATIONS, constants should be the same rules as other identifiers:
    public const int BoardLeftLocation = 100;

● Anti-rule 2:
  – Avoid Hungarian notation.
  – Do not give class names a prefix such as the letter C:
    CSprite
Private Variables

- Do not use `mPinballStartX`
- Use Camel Case and maybe something else
  - Note VB is case insenitive
- What is more error prone?
- Vote?
  - No, but have an opinion.

Version 1:
```java
private int _pinballStartX
public int PinballStartX { get...
public Foo(int pinballStartX) {
  _pinballStartX = pinballStartX;
}
```

Version 2:
```java
private int m_pinballStartX
public int PinballStartX { get...
public Foo(int pinballStartX) {
  m_pinballStartX = pinballStartX;
}
```

Version 3:
```java
private int pinballStartX;
public int PinballStartX { get...
public Foo(int pinballStartX) {
  this.pinballStartX = pinballStartX;
}
```

Version 4:
```java
private int pinballStartX;
public int PinballStartX { get...
public Foo(int aPinballStartX) {
  pinballStartX = aPinballStartX;
```
Back to software process

- Overview of methods
Software process

◆ What is the development strategy that we have surveyed so far?
  – Requirements elicitation – use cases
  – Decide on some architectural requirements – behavior diagrams
  – Create design – class diagram
    » Note – architecture and design work hand in hand
  – These steps are what you have done before:
    » Implement / Test
    » Deploy

◆ Very waterfall like
  – See next for review of waterfall
  – But incremental is the defacto process, we just haven’t had the time to do a real incremental approach
Classic “waterfall” lifecycle

System engineering

Analysis

Design

Coding

Testing

Maintenance

What

How

Change
What about processes that support development?

- Not all development tasks fit neatly into the waterfall model.
- Where do these tasks fit?
Support of Software Engineering

Planning, estimation, organization, process, management, ...

- System engineering
- Analysis
- Design
- Coding
- Testing
- Maintenance

Configuration control, metrics, defect tracking, reuse, documentation, ...
Pros and Cons of Waterfall

- Was used the most (and still used some)
- Projects are rarely sequential, thus change causes problems
- Difficult to get all customer requirements up front
- Customer does not see progress since first version is way late in process
- Early mistake can be disastrous
- Time waiting can exceed time being productive
- Significantly better than no process
Traditional/CMM vs. Agility

- **Capability Maturity Model**
  - Late 80’s, early 90’s
  - Thousands have adopted
  - Highly structured version of traditional software development

- **Has 5 levels**
  - Ad-hoc, chaotic
  - Repeatable
  - Defined
  - Managed
  - Optimizing

- **Complaints**
  - Spending time writing docs
  - Changing so fast that repeatable is expensive

- **Agility Manifesto**
  "We have come to value:

<table>
<thead>
<tr>
<th>Agile</th>
<th>Traditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals and Interactions</td>
<td>Process and tools</td>
</tr>
<tr>
<td>Working software</td>
<td>Comprehensive documentation</td>
</tr>
<tr>
<td>Customer collaboration</td>
<td>Contract negotiation</td>
</tr>
<tr>
<td>Responding to change</td>
<td>Following a plan</td>
</tr>
</tbody>
</table>

- That is, while there is value in the items on the right, we value the items on the left more.”

- **Complaints**
  - Can you do big projects with no documentation?
  - Isn’t this just the hack approach?
eXtreme Programming – XP

- One type of agile process that received a lot of support
- Write user stories on cards
- Define acceptance tests
- Estimate time to complete stories / Customer prioritize
- Define iteration/releases – highly incremental process
- Individual owns a story, works in pairs doing test-first development
- Collective ownership, simple design, continuous integration, 40 hour work week
- Releases of “business value” every few weeks to customer
Clearly XP Has “Made It Big” 😊

EXTREME PROGRAMMING
I CAN’T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.

AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A “USER STORY.”

OKAY, HERE’S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I’LL RUIN YOUR LIFE.
WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.

FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.

THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.

EXTREME PROGRAMMING

THE TWO OF YOU WILL BE A CODE-WRITING TEAM.

STUDIES PROVE THAT TWO PROGRAMMERS ON ONE COMPUTER IS THE MOST PRODUC TIVE ARRANGEMENT.

SOMETIMES I CAN WHISTLE THROUGH BOTH NOSTRILS. I'VE SAVED A FORTUNE IN HARMONICAS.
Crystal Clear

- Another type of agile process
  - Although trying to specify more general concepts
- “Crystal Clear – Human Powered Methodology for Small Teams”, Alistair Cockburn, Addison-Wesley, 2005
- … “is a lucid and practical introduction to running a successful agile project in your organization.”
- This portion comes from the “Crystal Clear” book
Seven Properties of Successful Teams

- Idea is to encourage the properties and not focus on the procedures
  - Because it is possible that other procedures can produce the same properties

1. Frequent Delivery
2. Reflective Improvement
3. Osmotic Communication
4. Personal Safety
5. Focus
6. Easy Access to Expert Users
7. Technical Environment with Automated Tests, Configuration Management, and Frequent Integration
1. Frequent Delivery

- **Why?**
  - Feedback on rate of progress
  - Users can give feedback on what can and can’t be done
  - Developers are forced into meeting short deadlines
  - Team can debug development and deployment process

- **How often?**
  - Two months
  - As long as four, and as little as weekly

- **What if can’t give to all?**
  - Find a test user and give it to them
  - If not, then at least integrate and test (doesn’t test deployment)
2. Reflective Improvement

- Meet and discuss what works and what doesn’t
  - AND FIX THOSE THINGS THAT AREN’T WORKING!!!
- Teams can do this and salvage a failing project
  - Lots of examples where a project is failing in the beginning
  - Lots of examples in the book where this led to success
- Try new ideas
- Post these lists so everyone can see
  - That way you don’t fall into the same traps
  - It also enforces the things that are working well
3. Osmotic Communication

- Communicate in the background so you can hear things as if by “osmosis”
  - That means that you just hear stuff that could be useful
- You can’t do this with everyone sitting in their own offices
  - Sit in the same room with the ability to just ask a question and get an answer
- You need to develop the skill to ignore questions that are not relevant
  - Supposedly this can be “learned”
- So – communication cost is low; feedback is high; errors are quickly fixed; knowledge is transferred
3. Osmotic Communication - 2

- **Problem** – too many questions, too much noise
  - Self regulates
  - Reduces random chatter
  - People respect “think time”

- **Problem** – protect the “project lead” by placing in their own office (they are in high demand)
  - Bad – can’t get questions answers
  - Bad – can’t learn (apprentice)

- **Solution** – put “project lead” in with everyone else
  - Can be bad – too many interruptions, lead can’t get anything done
  - Solution – develop “Cone of Silence”
4. Personal Safety

- Ability to speak when something is bothering you
  - Without fear of bad consequences
  - i.e. Your design needs to be improved
  - i.e. Mr. Manager, your schedule is unrealistic
  - i.e. Buddy, take a shower

- Is on the road to trust

- Presence of trust is directly related to teams that are successful
  - Personal observation – trust and dedication are worth more than just about anything for a team
    » In my experience a tiny team that trusts and is dedicated can beat everyone!!!
4. Personal Safety - 2

- Exposures
  - Revealing one’s ignorance
  - Revealing a mistake
  - Revealing ones inability to handle an assignment

- If you are exposed and nothing bad happens, then you build trust

- Some expert managers force an exposure, handle it in an appropriate way and demonstrate that you aren’t hurt by it
  - Building trust
5. Focus

1) Know what to work on, and
2) Have the time to do it

- Get someone who knows the business value to decide what is important
- Don’t work on several projects at once
  - Can’t progress on any
  - One to one and a half is the max

- Lots of interruptions
  - Begin to idle between interruptions and get nothing done
  - A solution is to block out two hours per day when only the very highest priority interruptions are allowed
    » No meetings, no demos, etc.
6. Easy Access to Expert Users

- This gives you:
  - Tester of your deliveries
  - Feedback on quality
  - Feedback on design decisions
  - The latest requirements
- The more hours an expert is available, the better
  - But the first hour is critical
- Success – at least weekly or biweekly meetings
  - Phone calls for backup
- On-site if possible
- Lastly try to get the developers to be trained as users
7. Technical Environment with Automated Tests, Configuration Management, and Frequent Integration

- Every programmer Alistair interviewed: “Once moved to automated tests swore never to work without them again.”
  - Automated tests improve quality of life
    » When you pass those tests on Friday afternoon, your weekend is free from worry

- Configuration management allows:
  - Working asynchronously
  - Backout changes
  - Create releases …

- Frequent Integration
  - Multiple times per day or at worst every other day

- Proposed best – continuous integration with test
Coming up…

- Unit testing in C#