Care about your craft notes

**Intro**

Who am I?

Why am I here?

How long have I been doing this?

**Quick slide**

Professionally, I work for Quick Solutions, even though none of these people do.

Been with Quick for four years, all in the App Dev Group started by BHP.

Been developing for over 10 years, all in Columbus.

**Nash Slide**

**Ask Why**

Why should I use this software?

Why should I use this language?

Why this framework?

Why do users do it this way?

**Everything is a People Problem**

Work with users as close as you can. Provide them options if possible.

Work with colleagues.

Damage control when the time comes – have a solution.

**You don’t know more than the users.**

“You don’t know what you want, Luigi knows what you want.”

You don’t know the process better than the users.

Provide value where you can improve their process…ask, “Why?”

Beware of, “That’s the way it’s always been,” answers. Dig deeper.

Example: Monkey story

**Communication.**

Daily Stand-up

Project Wiki

Email, IM, Twitter

Clear, professional emails with users (5 sentence rule)

Example: Aaron saying, “Oh, we’ve done that already,” after I solved a difficult problem on M1

Notify teammates of refactorings.

**Justify tech use.**

TFS v. Subversion

Critically analyze any technology/framework/language – (Microsoft “sheep” issue)

Example: “Vote of no confidence” on EF

Don’t use new just because it’s new

🡪 Move to “design”

**Design By Contract.**

Use interfaces

Method requirements in parameters, exception if they’re incorrect

Crash Early

**Don’t Program by Coincidence**

Test, test, test

Prove your design

Example: FU professor bringing down 800 service at at&t

Example: Trial and error…first two trials work, but when error shows up you don’t know why because you never understood how it worked in the first place.

**Don’t be a plumber**

Use frameworks – nHibernate, LINQ to SQL, Ninject, Windsor, MVC

**You ain’t gonna need it.**

Resist adding code that might help in the future.

TDD helps stop this.

**Keep an eye on the big picture**

Feature drift/scope creep

Know what other members of your team are working on (Stand-up, communication)

Explain stand-up meeting: 13 minutes, what you did yesterday, what you’re doing today, what’s holding you up

🡪 Move on to coding practices

**Quality**

“At Ford, Quality is Job 1.”

Make quality a requirements issue.

Nobody sets out to write bad code, but bad habits form.

“Everybody else is doing it, man!”

**DRY**

Don’t Repeat Yourself

Once is initial code, twice is a coincidence, third time is a pattern.

Example: RPS Trailer moving, worked hard not to move the same one twice. Didn’t follow that as much when I got to software.

“Copy, paste, 10 minutes.”

Think about what you’re doing

Orthagonality

Borrowed from geometry, means moving along one line will not change your position on an intersecting line. Such as points on an x/y axis.

Software it means making changes to one part of the system will not effect the others, like changing your db shouldn’t impact the UI

Terms:

1. Decoupled
2. Seperation of Concerns

MVC

Business Logic layer adding UI items to a select list. (Putting “Select” in when it’s a UI only option.)

🡪 Move into Testing

**Unit Testing**

TDD/BDD

1. Red, Green, Refactor
2. Test one thing per test
3. Use mocking to get unrelated things out of the way
4. Write testable code

**Test your software, your users will**

Beyond unit tests

1. Integration tests
2. System tests
3. Performance tests

PP refers to it as “Ruthless testing.”

Example: IBM “Black Team” from Peopleware. – emphasize the culture created here

🡪 Move to automated processes

**Automate.**

Automate, don’t do processes manually.

Lots fall into this – code generation, use of source control, start up procedures, macros, etc

Specifically here, Build process

Use continuous integration – TFS, CC.Net, nAnt, msBuild, Team City, etc

Automate the running of your unit tests

Failed unit test == broken build

Don’t live with a broken build

Don’t check in code when the build is broken

Examples:

1. ThoughtWorks example of client commenting out the tests
2. Todd’s example of the three month broken build
3. Our 5 day broken build on VCS

**Collective Ownership**

“It takes a village” theory

Don’t refer to problems as “yours” but rather “ours”

Don’t pull out the blamethrower, don’t be a blameosaurus

Everybody takes responsibility for a broken build.

1. Stop everything when build breaks
2. Determine source of break
3. Breaker should fix the build, or at least be made aware of what broke it. (Communication!)

Example: Josh’s dunce cap example with the lava lamp, our lunch buying example

**Don’t live with broken windows**

Refactor them out immediately

Example: CSS on large projects

Example: Get v. Load in nH on M1 which lead to a programmatic catch block

**Vista Slide**

**Refactoring**

Refactor early and often

Why do we refactor?

1. Remove broken windows
2. Found a better way to do something through new knowledge
3. Improve performance
4. You found some wet code
5. You’ve found some tightly coupled code, non-orthagonal, like the page\_load in a web form.

**Good Enough.**

Know when to say when.

Avoid gold plating, adding value beyond the specs.

Avoid Refactorbation, refactoring code for your own pleasure and no real gain.

You can’t write perfect software.

🡪 Move into debugging

**Debugging.**

Find bugs once and only once

Wrap with a unit test

Hardest problems first – Finz example of doing low levels first to “get more done.”

Time box difficult problems

1. Give yourself a reasonable amount of time
2. Keeps from floundering or spinning of wheels
3. Keeps momentum going

Rubber Ducking

1. Second set of eyes, bobbing head “like a rubber duck.”
2. Explain the problem

Example: Alexei walking into my cube, I thought what I was going to tell him and found the issue

**Don’t panic**

Don’t rush

Yes, there’s a problem, rushing through it may create more

Slow down, THINK about the problem

Don’t assume it, prove it

Isolate the code that is causing the bug.

Remove outside code from equation – db isn’t the problem, framework isn’t problem

Example 1: Phil working with Ent Lib, debugger stopped in Ent Lib so he went there rather than into his code that was calling it

Example 2: Finz with the VAB. VAB throwing object error, he was ready to throw his hands up and remove it. Issue was he was passing a null object to it.

**Fix the problem, not the blame**

Put away the blamethrower, finding out who did it doesn’t get the issue solved any quicker

Example: PM having me search down who and when an issue was created, took me four hours when the fix only took one.

Provide Solutions, not excuses

Take responsibility for issues

If you find a problem, have a solution before taking it up the ladder

🡪 Move into continuous improvement

**Continuous Improvement**

Have a mentor

Learn a new language: my choice is Ruby

Certs, blogs, books, etc.

Know what to unlearn

“That’s the way it’s always been.” – Old habits die hard. (Refer to don’t be a plumber.)

Be a Specializing Generalist - Jim Holmes

No Best Practices, only better practices – we’re hung up on the Best Practices phrase, but they’re constantly evolving

**Keep your toolbox full.**

Know your IDE

Add plug-ins to help productivity (Slick Edit tools, Resharper, Code Rush/Refactor)

Lower your mouse dependency (Move mouse to the other side of the keyboard)

Get to know a good text editor (Notepad2, Notepad ++, E)

Good source control.

**Attitude.**

5:01 developers – OK, but help them along in their 8 hours

Be a catalyst for change

“Negativity kills ingenuity.” – T. Kaufman butchering a Venkat saying

**Be a Mentor.**

Have a mentor, but also BE a mentor

“Eat like a bird, poop like an elephant” – Guy Kawasaki

Lunch and Learns

Book reviews

Allow people to figure it out. Point them in the right direction, but don’t solve it for them.

**Create a culture**

Being a catalyst for change and being a mentor, create a culture

Make TDD, CI, and positive attitudes habits

Adopt the stand-up meeting, even if you’re not full-on agile

Example: “Donuts!”

Reference the IBM Black team again

**End**