Working With Third Party Code

Thaddaeus Frogley

Who Am I

- Thaddaeus Frogley
  Senior Programmer, Climax
- Over 10 years games industry experience, shipped ports, sequels, and original titles

invited by nordmedia + northstar developers

What Is 3rd Party Code

- Not written by your team
- aka "Legacy Code"
- Supplied by other people / organisations

Outside your control

"legacy code" implies out of date and abandoned -> 3rd party code CAN be legacy code, but could also be code actively in development

You and your team have to work with, perhaps even modify

Examples

- Middleware
- Ports & sequels
- Core tech

Core Tech or Shared Code made by other game teams in your studio

3rd Party code can be any code your team didn't write

Why it is important

- Middleware can save time and reduce risk
- Projects are only 1/100 to 1/1000 new code

worth discussing because: We all have to do it, it is the major body of work that is done by developers

even if you develop own-tech, some of the code will be 3rd party

Michael Feathers – “In many new development
efforts, the amount of legacy code will overwhelm the amount of new code by factors of 100 to 1, or 1000 to 1.” (Working Effectively With Legacy Code)

Why it is important

worth discussing because: We all have to do it, it is the major body of work that is done by developers

even if you develop own-tech, some of the code will be 3rd party

Michael Feathers - “In many new development
efforts, the amount of legacy code will overwhelm the amount of new code by factors of 100 to 1, or 1000 to 1.” (Working Effectively With Legacy Code)

Why it is important

don’t talk about it because:

It’s new technology that grabs the headlines. It’s not glamorous.

As an industry the attitude is often: build it, ship it, and forget about it. All too often issues of “maintenance” are neglected.

Why it is important

- Outside the games industry this work is called "Software Maintenance"
- IEEE International Conference on Software Maintenance - http://icsm07.ai.univ-paris8.fr/

don't talk about it because:

It's new technology that grabs the headlines. It's not glamorous.

As an industry the attitude is often: build it, ship it, and forget about it. All too often issues of "maintenance" are neglected.

Why it is important

don't talk about it because:

It's new technology that grabs the headlines. It's not glamorous.

As an industry the attitude is often: build it, ship it, and forget about it. All too often issues of "maintenance" are neglected.

Why it is important

IEEE describes Maintenance like so: ...

This describes exactly the kind of work that **games** programmers spend the majority of their day doing.

But WHY? Because it is difficult...

Why it is important

- "Maintenance extends from correction of code to adaptation, and enhancement of systems, designs and architectures."

IEEE describes Maintenance like so: ...

This describes exactly the kind of work that **games** programmers spend the majority of their day doing.

But WHY? Because it is difficult...

Why it is difficult

- PS2/PSP Game: 470K LOC
- Xbox360 game: 850K LOC
- New-gen middleware: 1580K LOC
- "One full-time maintenance person for every 20K LOC" - Thomas Pigoski

Game code-bases are quite big - and getting bigger. (from almost 500 thousand lines to over one and a half million lines of code) - I even heard one programmer claim his projects library topped 2.8M LOC

Pigoski -> Software Contractor, US Military, wrote the book “Practical Software Maintenance” : Frame of Reference.

Note: 1 per 20k is not a recommendation, games typically use more like 1 per 100k ...

gives you an idea of the scale of the task - the sheer volume of information dealt with

Why it is difficult

Game code-bases are quite big - and getting bigger. (from almost 500 thousand lines to over one and a half million lines of code) - I even heard one programmer claim his projects library topped 2.8M LOC

Pigoski -> Software Contractor, US Military, wrote the book “Practical Software Maintenance” : Frame of Reference.

Note: 1 per 20k is not a recommendation, games typically use more like 1 per 100k ...

gives you an idea of the scale of the task - the sheer volume of information dealt with

Why it is difficult

get a programmer talking over a few beers and with a little encouragement you can probably get it out of him that he thinks ...

some examples to demonstrate why programmers end up feeling that way ....

This example shows good intentions... the code the follows is NOT used in a specific bug fix

... but ...

... bug ids are useful, but need to be accompanied by descriptive comments as well

Why it is difficult

- Other people's code "sucks"

get a programmer talking over a few beers and with a little encouragement you can probably get it out of him that he thinks ...

some examples to demonstrate why programmers end up feeling that way ....

This example shows good intentions... the code the follows is NOT used in a specific bug fix

... but ...

... bug ids are useful, but need to be accompanied by descriptive comments as well

Why it is difficult

- Other people’s code “sucks”

```
#if !WorkaroundBug7084
```

get a programmer talking over a few beers and with a little encouragement you can probably get it out of him that he thinks ...

some examples to demonstrate why programmers end up feeling that way ....

This example shows good intentions... the code the follows is NOT used in a specific bug fix

... but ...

... bug ids are useful, but need to be accompanied by descriptive comments as well

Why it is difficult

- Other people's code "sucks"

```
#if !WorkaroundBug7084
```

- Which could be useful… if you had access to their bug database

get a programmer talking over a few beers and with a little encouragement you can probably get it out of him that he thinks ...

some examples to demonstrate why programmers end up feeling that way ....

This example shows good intentions... the code the follows is NOT used in a specific bug fix

... but ...

... bug ids are useful, but need to be accompanied by descriptive comments as well

Why it is difficult

- Other people's code "sucks"

get a programmer talking over a few beers and with a little encouragement you can probably get it out of him that he thinks ...

some examples to demonstrate why programmers end up feeling that way ....

This example shows good intentions... the code the follows is NOT used in a specific bug fix

... but ...

... bug ids are useful, but need to be accompanied by descriptive comments as well

Why it is difficult

- Other people’s code “sucks”

Why it is difficult

- Other people’s code “sucks”

```
sceneMainIsPausedOrAboutToPauseButNotAboutToPlay(uni
t->scene)
```

Why it is difficult

- Other people's code "sucks"

```
sceneMainIsPausedOrAboutToPauseButNotAboutToPlay(uni
t->scene)
```

- What does the function mean? Where are the "brackets"?

Why it is difficult

- Other people's code "sucks"

Why it is difficult

- Other people's code "sucks"

can be good for moral to have a laugh ... intellectual laziness ... need to put it in context, and understand WHY the code is like that...

but before i go on to tips / best practices section, lets look at some general pitfalls

Why it is difficult

- Other people's code "sucks"

```
mpLayer = mpScreen-
>AddLayer(FECK2::CLayer::Create().Name("..."
).Depth(10000).EventDeclaration(new
CLEFunctor(this,&FUNCTION_NAME(CFENonmodal,
EE_DECLARE_HANDLERS))));
```

can be good for moral to have a laugh ... intellectual laziness ... need to put it in context, and understand WHY the code is like that...

but before i go on to tips / best practices section, lets look at some general pitfalls

Why it is difficult

- Other people’s code “sucks”

```
mpLayer = mpScreen-
>AddLayer(FECK2::CLayer::Create().Name("..."
).Depth(10000).EventDeclaration(new
CLEFunctor(this,&FUNCTION_NAME(CFENonmodal,
EE_DECLARE_HANDLERS))));
```

- Funny ... But its the wrong attitude!

can be good for moral to have a laugh ... intellectual laziness ... need to put it in context, and understand WHY the code is like that...

but before i go on to tips / best practices section, lets look at some general pitfalls

Pitfalls

- Poorly named variables
- Out dated comments

Can’t take code comments or variable names as face value

Results in: Misinterpreting Intent - can mislead or distract

where code & comments contradict, how can you tell which is “right”
(buggy code or out of date comment?)

Pitfalls

Can’t take code comments or variable names as face value

Results in: Misinterpreting Intent - can mislead or distract

where code & comments contradict, how can you tell which is “right”
(buggy code or out of date comment?)

Pitfalls

functional relationships - undocumented - not enforced in code structure
Usually in the form: A and B must run in that order, or C doesn't work properly

means that apparently simple changes (bug fixes) can break unexpected things

Decoy code doesn't get used, but may be compiled & linked. Not isolated, but mixed up with "live code".
Results in: Time wasted trying to understand how dead code works.

Pitfalls

- Implicit dependancies
- Decoy code

functional relationships - undocumented - not enforced in code structure
Usually in the form: A and B must run in that order, or C doesn't work properly

means that apparently simple changes (bug fixes) can break unexpected things

Decoy code doesn't get used, but may be compiled & linked. Not isolated, but mixed up with "live code".
Results in: Time wasted trying to understand how dead code works.

Pitfalls

functional relationships - undocumented - not enforced in code structure
Usually in the form: A and B must run in that order, or C doesn’t work properly

means that apparently simple changes (bug fixes) can break unexpected things

Decoy code doesn’t get used, but may be compiled & linked. Not isolated, but mixed up with “live code”.
Results in: Time wasted trying to understand how dead code works.

Pitfalls

Working with systems you didn't write, you don't fully understand the code ... taking for granted that something that looks wrong is wrong can result in mistakes of your own.

There is a body of knowledge build up by a development team that exists only as their experience, no matter how well a team documents or comments their code, they cannot pass on their all private idioms and attitudes.

Pitfalls

- You don’t know all the forces that shaped the design decisions
- Separating the code that is broken, and needs to be fixed from the code that is simply not fully understood yet can be nontrivial

Working with systems you didn’t write, you don’t fully understand the code ... taking for granted that something that looks wrong is wrong can result in mistakes of your own.

There is a body of knowledge build up by a development team that exists only as their experience, no matter how well a team documents or comments their code, they cannot pass on their all private idioms and attitudes.

Pitfalls

Working with systems you didn’t write, you don’t fully understand the code ... taking for granted that something that looks wrong is wrong can result in mistakes of your own.

There is a body of knowledge build up by a development team that exists only as their experience, no matter how well a team documents or comments their code, they cannot pass on their all private idioms and attitudes.

Pitfalls

...
The teams ability to work with the code will continue to improve the longer they work with it

Next: Best practices -> how to best mitigate the impact of these pitfalls on your project

Pitfalls

- Changes risk introducing new problems
- Takes time for you to “ramp up” to full production

...
The teams ability to work with the code will continue to improve the longer they work with it

Next: Best practices -> how to best mitigate the impact of these pitfalls on your project

Best Practices

- Try to have a hand-over meeting & social event with key team members before kick off
- Establish a good rapport

Building a relationship with the 3rd party, as either customer or client

in the case of Middleware - evaluated before you use on a project - a meeting can be a useful part of that evaluation

ports/sequels OTOH the deal may done before you see the code

Understanding them helps you to understand their code, no better way to understand someone than sit down and talk to them

Best Practices

Building a relationship with the 3rd party, as either customer or client

in the case of Middleware - evaluated before you use on a project - a meeting can be a useful part of that evaluation

ports/sequels OTOH the deal may done before you see the code

Understanding them helps you to understand their code, no better way to understand someone than sit down and talk to them

Best Practices

ranked list, best to worst, for quality of communication

quality of contact vs quantity of contact -> infrequent, high quality exchanges are of more value than constant stream of noise

use friendly & diplomatic staff members as your contacts with externals

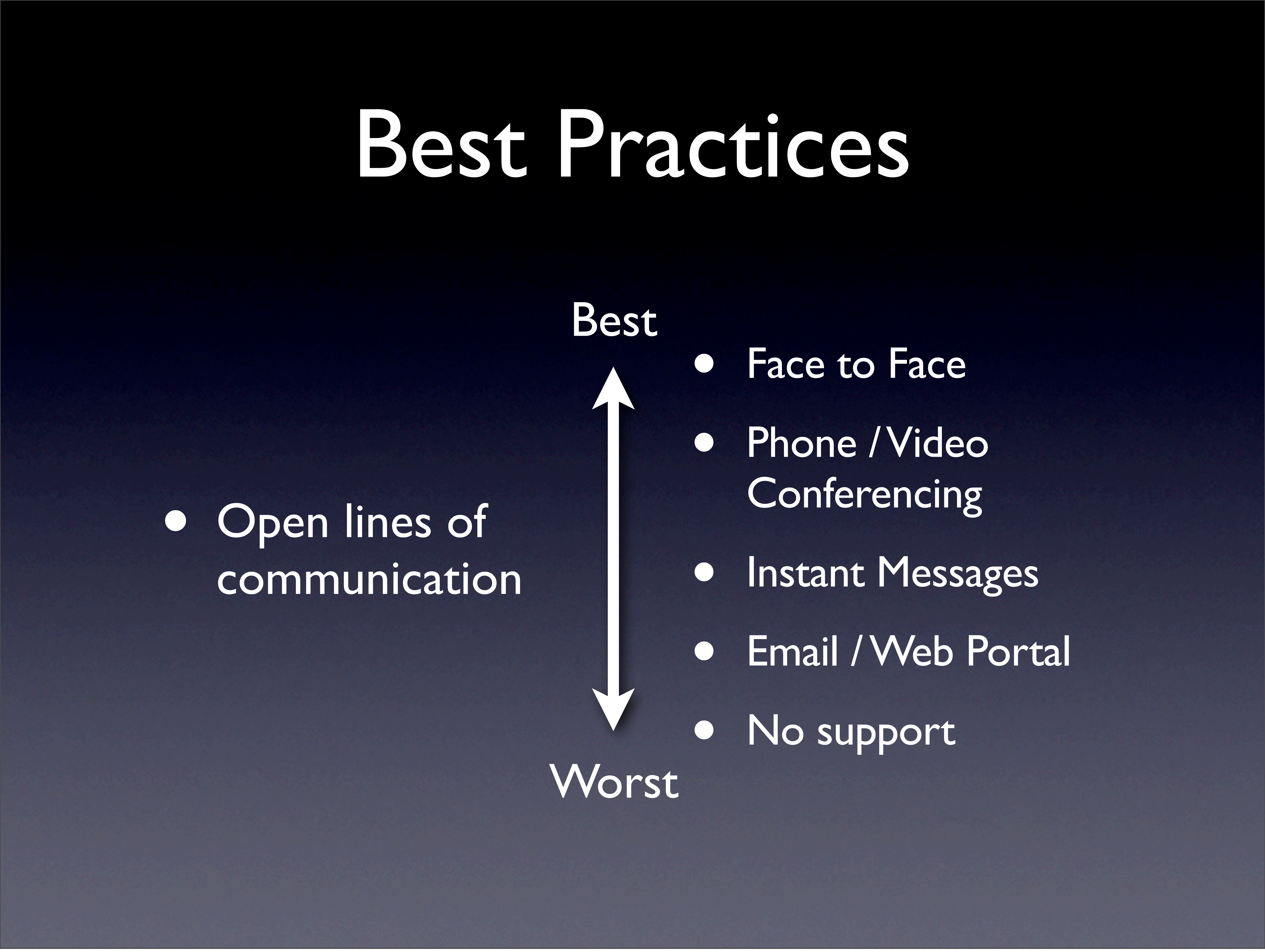
Best Practices

- Open lines of communication

ranked list, best to worst, for quality of communication

quality of contact vs quantity of contact -> infrequent, high quality exchanges are of more value than constant stream of noise

use friendly & diplomatic staff members as your contacts with externals



Best Practices

- Open lines of communication

Best

- Face to Face
- Phone / Video Conferencing
- Instant Messages
- Email / Web Portal
- No support

Worst

ranked list, best to worst, for quality of communication

quality of contact vs quantity of contact -> infrequent, high quality exchanges are of more value than constant stream of noise

use friendly & diplomatic staff members as your contacts with externals

Best Practices

ranked list, best to worst, for quality of communication

quality of contact vs quantity of contact -> infrequent, high quality exchanges are of more value than constant stream of noise

use friendly & diplomatic staff members as your contacts with externals

Best Practices

(Games)

a large number of defects come from people “not knowing” about a feature

1 game expert

~ 1 weeks worth of time scheduled at the start of the project

Best Practices

- Get to know the original product
- Play the original game in depth

(Games)

a large number of defects come from people “not knowing” about a feature

1 game expert

~ 1 weeks worth of time scheduled at the start of the project

Best Practices

- Get to know the original product

(Games)

a large number of defects come from people “not knowing” about a feature

1 game expert

~ 1 weeks worth of time scheduled at the start of the project

Best Practices

- Get to know the original product

(Middleware)

Check out games made by other companies using that middleware.

How did they do what they did? Is there a common theme in what have they not done?

Are there common problems or flaws? Spot that and figure out why, and you can save your team a lot of wasted time.

Next: RCV

Best Practices

- Get to know the original product

- Read the documentation
- Read the mailing lists
- Look at the examples

(Middleware)

Check out games made by other companies using that middleware.

How did they do what they did? Is there a common theme in what have they not done?

Are there common problems or flaws? Spot that and figure out why, and you can save your team a lot of wasted time.

Next: RCV

Best Practices

(Middleware)

Check out games made by other companies using that middleware.

How did they do what they did? Is there a common theme in what have they not done?

Are there common problems or flaws? Spot that and figure out why, and you can save your team a lot of wasted time.

Next: RCV

Best Practices

Undo Button for your whole project. It’s not just a backup, but a powerful tool for managing change, division of workload, and isolating risk

asset control as well - ie texture file location in one place, not spread around a whole art departments hard drives

if a commercial solution isn’t for you, alternatives also exist

Best Practices

- Revision Control Systems
- Commercial: Perforce, Alienbrain
- Open source: CVS, Subversion

Undo Button for your whole project. It’s not just a backup, but a powerful tool for managing change, division of workload, and isolating risk

asset control as well - ie texture file location in one place, not spread around a whole art departments hard drives

if a commercial solution isn’t for you, alternatives also exist

Best Practices

Undo Button for your whole project. It's not just a backup, but a powerful tool for managing change, division of workload, and isolating risk

asset control as well - ie texture file location in one place, not spread around a whole art departments hard drives

if a commercial solution isn't for you, alternatives also exist

Best Practices

A retail copy - QA can compare to original

Debuggable reference build -> Mitigate the dead code issue by being able to debug the clean vendor version of the codebase

Browse-able (read only) copy of the original code-base - essential! As version zero, or a RCS branch!

Best Practices

- Reference build
- Reference code

A retail copy - QA can compare to original

Debuggable reference build -> Mitigate the dead code issue by being able to debug the clean vendor version of the codebase

Browse-able (read only) copy of the original code-base - essential! As version zero, or a RCS branch!

Best Practices

A retail copy - QA can compare to original

Debuggable reference build -> Mitigate the dead code issue by being able to debug the clean vendor version of the codebase

Browse-able (read only) copy of the original code-base - essential! As version zero, or a RCS branch!

Best Practices

I would recommend the team keeps wiki page, to spread knowledge amongst them.

Code reviews spread knowledge and also reinforce and verify coding rules & guidelines (one of which I'll talk about on the next slide) – are being followed by the whole team. There *is* time to review *every* change if you spread the load…

Next Slide: Tech Issues

Best Practices

- Internal documentation
- Code reviews

I would recommend the team keeps wiki page, to spread knowledge amongst them.

Code reviews spread knowledge and also reinforce and verify coding rules & guidelines (one of which I'll talk about on the next slide) – are being followed by the whole team. There *is* time to review *every* change if you spread the load…

Next Slide: Tech Issues

Tech issues

- Maintain separation
- Comment changes and insertion points distinctively

Importance of being able to quickly identify what parts of the codebase have been written/changed by your team, vs what is part of the original codebase – allows a mental gear change, results in quicker bug fix turn around.

Tech issues

- Maintain separation
- Comment changes and insertion points distinctively
- `// cx: (name) what changed and why`

Importance of being able to quickly identify what parts of the codebase have been written/changed by your team, vs what is part of the original codebase – allows a mental gear change, results in quicker bug fix turn around.

Tech issues

- Maintain separation
- Comment changes and insertion points distinctively
- `// cx: (name) what changed and why`
- Limit changes in the original code source files to inline changes
- Put functionality in new files, with a distinctive naming convention

Importance of being able to quickly identify what parts of the codebase have been written/changed by your team, vs what is part of the original codebase – allows a mental gear change, results in quicker bug fix turn around.

Tech issues

Importance of being able to quickly identify what parts of the codebase have been written/changed by your team, vs what is part of the original codebase - allows a mental gear change, results in quicker bug fix turn around.

Tech issues

Team decision -- Making a decision early and sticking to it is more important than what the decision is.

Issue to consider:
* Does the 3rd party code have a consistent style?
* New code reuse within the studio (use studio style).
* Do you have a strong in house style?

Tech issues

- Code style - Stick to your in house style, or switch to the 3rd party code style?

Team decision -- Making a decision early and sticking to it is more important than what the decision is.

Issue to consider:
* Does the 3rd party code have a consistent style?
* New code reuse within the studio (use studio style).
* Do you have a strong in house style?

Tech issues

Team decision -- Making a decision early and sticking to it is more important than what the decision is.

Issue to consider:
* Does the 3rd party code have a consistent style?
* New code reuse within the studio (use studio style).
* Do you have a strong in house style?

Tech issues

laudable aims, but can be a huge tasks to achieve in an old code-base, and though the changes required are low risk there can be a lot of them and it could be very time consuming.

fixing warnings can find bugs -> but (true story) 1400 warnings in 1 day could be a lot of code changes (or 1 header change)

Evaluate the workload before committing to this!

Tech issues

- Avoid cascading refactoring
- const-correct
- Warning-free

laudable aims, but can be a huge tasks to achieve in an old code-base, and though the changes required are low risk there can be a lot of them and it could be very time consuming.

fixing warnings can find bugs -> but (true story) 1400 warnings in 1 day could be a lot of code changes (or 1 header change)

Evaluate the workload before committing to this!

Tech issues

laudable aims, but can be a huge tasks to achieve in an old code-base, and though the changes required are low risk there can be a lot of them and it could be very time consuming.

fixing warnings can find bugs -> but (true story) 1400 warnings in 1 day could be a lot of code changes (or 1 header change)

Evaluate the workload before committing to this!

Tech issues

When you start working with a body of code you are sure to find design decisions you don't agree with.

A common point of contention is resource management. In my experience adapting to work with the architecture provided saves time overall. Changing core systems may make make certain changes easier on the surface, but you'll most likely end up having to solve problems the original team already dealt with

Where feasible build in compile-time or run time switches to switch off new code, so the original behaviour can be tests/compared against (after all, the publisher might change their minds!)

Tech issues

- Work with, not against, the architecture of the system
- When building foundations for new code, maintain the behaviour of existing code

When you start working with a body of code you are sure to find design decisions you don't agree with.

A common point of contention is resource management. In my experience adapting to work with the architecture provided saves time overall. Changing core systems may make make certain changes easier on the surface, but you'll most likely end up having to solve problems the original team already dealt with

Where feasible build in compile-time or run time switches to switch off new code, so the original behaviour can be tests/compared against (after all, the publisher might change their minds!)

Merging Vendor Drops

- Potentially time consuming & risky
- Use a branching RCS
- Use a good 3-way merge tool

When: Live ports, patched games, middleware updates.

Build on foundations of good practices based on Tracking Changes.

This may take several days, and could be worked on by several people – so should be done in it’s own branch

Whitepaper on P4 website

Bug Fixing

- Is this a bug in our changes, or a bug in the original game?
- Most bugs come from incorrect assumptions about what the code was supposed to do
- Programmers need to dig deeper

Use the reference build!

Try to find the source bug, and not just fix the symptom.

Due Diligence

- Unit tests
- Pair programming
- Continuous integration

Unit testing -> the life cycle of a game project isn't long enough to recoup the investment of time spent getting a project "under test", and unit tests don't cover the "soft" issues (such as graphics looking "good" or the game being "fun").

Pair programming -> Has it's uses in game development, and in projects using 3rd party code can be useful to spread knowledge, especially if you have a vendor coder on site for a while, but I have my doubts that strict PP methodogies work well for game dev

CI -> Build machine, more than daily build, needs extra hardware and licences, but means you have a "latest" build available at all times (producers can check this without delays or prgrammer time wasted), and catches bad check-ins quickly, without wasting a whole code teams time on 1 programmers mistake.

Conclusion

- Communication
- Track what you change
- Approach with caution
- Do not underestimate – or overestimate – the original developers
- Resist changing what you don't need to

Being able to identify at a glance and with confidence who wrote the code your looking at is a huge time saver.

Code can be subtle and complex in ways you'd never suspect.

There may be bugs, and there may be things that look like bugs on the surface, but are not.

Further Reading

- “Working Effectively with Legacy Code” (Robert C. Martin Series) by Michael Feathers
- “Refactoring: Improving the Design of Existing Code” by Martin Fowler et al.
- “Practical Software Maintenance” by Thomas Pigoski

These are the main software engineering textbooks on this topic.

They are written by experts in their fields, though they aren't written by game developers so not all of the advice they give necessarily applies to the games industry.

Any questions?

Thanks for listening!