IMGD 3000 - Technical Game Development I: Iterative Development Techniques

by

Robert W. Lindeman

gogo@wpi.edu

Motivation

- The *last* thing you want to do is write critical code near the end of a project
  - Induces *huge* stress on the team
  - Introduces all kinds of *interesting bugs* that break working code
- Testing *always* gets cut in a crunch
  - Makes the problem *even worse!*
- Planning can help avoid writing critical code in alpha or beta phases

Wishes Versus Reality

- Most games you play are less/smaller than originally envisioned
    - Design was bigger than implementation
    - Implementation was bigger than what actually made it into the game
- How do we know when a game is "done"?

How Do We Estimate Progress?

- Example:
  - Jo is a programmer
  - She estimates it will take 10 days to implement a Smart Trap
  - She is 4 days into the implementation
  - Is the Smart Trap 40% complete?
    - We may not see it "snap shut" until day 9
  - Say she is good, and finishes in 8 days total
    - We are ahead!
  - Later, it is decided to add functionality to the Smart Trap (e.g., can trap larger objects)
    - This takes 4 days
  - Now we’re behind!

So, What’s the Point?

- Most things get revisited multiple times during development
  - Fix bugs, modify functionality, etc.
- The "40% done" estimate looks pretty sketchy…
- We need a way to account for time without driving a project into trouble (and into panic)

Incremental Delivery

- Milestones are good things!
  - They let us get things done
- Downside
  - If you miss one, people notice, and action is often taken
  - Especially management and production people

Incremental Delivery (cont.)

- Developer' s view
  - Milestones (or *plans* in general) are just ***best guesses*** for how the implementation will evolve
- Management' s view
  - Schedules are contracts with developers
  - Promising certain things at certain times
- These different views cause problems
  - Developers: Panic, pressure, long hours
  - Managers: Justification, financial pressure

Milestones

- Without milestones, work will not get done
- ***Unrealistic*** milestones mean the work will not get done on time, regardless of how financially important they are
- Managers need to know the estimates of the developers, and the key markers along the way
  - They need to plan their financial links accordingly

Milestones (cont.)

- External (used by managers) milestones are at a coarser granularity
  - Need to tie to publishers, etc.
- Internal (used by developers) milestones are at a finer granularity
  - Need to use among team members

Milestones (cont.)

- Think of the development plan as a blackbox
  - Managers have a specific "interface" to the box
    - `Give me the latest build`
    - `Give me the latest (high-level) schedule`
- Clearly, this is too simplistic/wishful thinking
  - Managers want to know more
- But it helps separate things better

Hidden Gems

- For many, if I can't see it, it is not important
  - AI takes time to build
  - Network balancing is an optimization
- Developers receive less "credit" for these than things that can be seen
- Good managers will probe deeper below the surface to see what is really going on
  - Requires technical ability (knowledge)

Iteration

- Make frequent (daily, weekly?) working builds
  - "We don’t go home Friday until a working build is checked in."
  - If management asks for the latest build, give them the one from last week
- Resist the desire to show the latest-and-greatest
  - People will always expect it, and it leads to unrealistic expectations

Internal Scheduling

- Given a detailed design document
  - Make a list of all objects (players, items, NPCs, environments, etc.) that need to be built
  - Mark each one as either
    - Core,
    - Required, or
    - Desired.
  - Remember the circle diagram?
- End result
  - List of features sorted by importance

Internal Schedule Structure

- Could start working from top of list, and when time runs out, we are done
  - Produces a lot of complete pieces, but no whole
  - Makes management (and others) nervous
- Since we made the list in an OO way, we should start building objects!

OO Iterative Development: Object Versions

- Create a *Null* version for each object
  - Complete, but empty
- *Basic* version
  - Placeholder with some properties present
- *Nominal* version
  - Commercially viable implementation
- *Optimal* version
  - State of the art version

```
// Player.h
class Player  {
  public:
     Player( void );
     ~Player( void );
};

//Player.cpp
#include "Player.h"


Player::Player( void )   {
}


Player::~Player( void )   {
}
```

OO Iterative Development: Object Versions (cont.)

- Some objects will be simpler
  - Fewer iterations
- Some will be more complex
  - More iterations
- We can say we have a ***shippable*** game when every object is at least at the *Nominal* version
- A ***complete*** game is one where all objects are at *Optimal* level

Discussion

- Seems like we need to write *three* versions of every object!
  - Yes, but we would probably do this anyway with revisions
- Approach
  - Starting with core, then required, then desired, implement *Null* versions of all objects
  - Starting with core, then required, implement the *Nominal* versions
    - Code is now *releasable*
  - Start to work on desirables

Discussion (cont.)

- This is a breadth-first approach
- Better than "let's do the cool bits first!"
  - Always have a build-able game
  - Near-continuous growth
  - Can easily show refinement
  - Better handle on how "complete" the game is

Scheduling: Naïve

	Feature	Null	Base	Nominal	Optimal
Core	F1	1	13	25	37
	F2	2	14	26	38
	F3	3	15	27	39
	F4	4	16	28	40
Required	F5	5	17	29	41
	F6	6	18	30	42
	F7	7	19	31	43
	F8	8	20	32	44
Desired	F9	9	21	33	45
	F10	10	22	34	46
	F11	11	23	35	47
	F12	12	24	36	48

Scheduling: Better (single programmer)

	Feature	Null	Base	Nominal	Optimal
Core	F1	1	13	22	37
	F2	2	14	23	38
	F3	3	15	24	39
	F4	4	16	25	40
Required	F5	5	17	26	41
	F6	6	18	27	42
	F7	7	19	28	43
	F8	8	20	29	44
Desired	F9	9	21	32	45
	F10	10	30	33	46
	F11	11	31	34	47
	F12	12	35	36	48

Scheduling:
Better (multiple programmers)

	Feature	Null	Base	Nominal	Optimal
Core	F1	1A	7A	11B	19A
	F2	1B	7B	12A	19B
	F3	2A	8A	12B	20A
	F4	2B	8B	13A	20B
Required	F5	3A	9A	13B	21A
	F6	3B	9B	14A	21B
	F7	4A	10A	14B	22A
	F8	4B	10B	15A	22B
Desired	F9	5A	11A	16B	23A
	F10	5B	15B	17A	23B
	F11	6A	16A	17B	24A
	F12	6B	18A	18B	24B

Team Utilization

- Make sure to use the skills of each team member well
  - All eggs in one basket
  - Jack of all traits, master of none
- Keep everyone busy
  - No waiting, if possible
- Communication is vital
  - Every programmer should be aware of what others are doing
    - Code reviews
    - Joint status meetings
    - Documentation

Scheduling:
Eggs in one Basket

	Feature	Null	Base	Nominal	Optimal
Core	F1	1A	7A	12A	19A
	F2	1B	7B	11B	19B
	F3	2A	8A	13A	20A
	F4	2B	8B	12B	20B
Required	F5	3A	9A	14A	21A
	F6	3B	9B	13B	21B
	F7	4A	10A	15A	22A
	F8	4B	10B	14B	22B
Desired	F9	5A	11A	16A	23A
	F10	5B	15B	16B	23B
	F11	6A	17A	18A	24A
	F12	6B	17B	18B	24B

Scheduling with Iteration

- Shift:
  - FROM: When will it be finished?
  - TO: When will it be good enough?
- "Finished" is meaningless anyway
- We have a definition of "Good Enough" now!
- Bad estimation often comes from top-down dissection
  - No accounting for the learning curve, code revision, or integration
- Iterative development
  - Total time equals the sum of the Null, Base, Nominal, and Optimal levels