CMPT 373
Software Development Methods

Agility, Refinement, & Integration

Nick Sumner
wsumner@sfu.ca

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - **1950s & 1960s** at NASA, driven by approaches from the **1940s**

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - 1950s & 1960s at NASA, driven by approaches from the 1940s
  - “Program development by stepwise refinement” - Wirth 1971

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - 1950s & 1960s at NASA, driven by approaches from the 1940s
  - “Program development by stepwise refinement” - Wirth 1971
- The focus of the increments was a little different than today

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - 1950s & 1960s at NASA, driven by approaches from the 1940s
  - “Program development by stepwise refinement” - Wirth 1971
- The focus of the increments was a little different than today
- And then software development as an industry boomed

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - 1950s & 1960s at NASA, driven by approaches from the 1940s
  - “Program development by stepwise refinement” - Wirth 1971
- The focus of the increments was a little different than today
- And then software development as an industry boomed
  - Companies focused on long term commitments

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - 1950s & 1960s at NASA, driven by approaches from the 1940s
  - "Program development by stepwise refinement" - Wirth 1971
- The focus of the increments was a little different than today
- And then software development as an industry boomed
  - Companies focused on long term commitments
  - Managers wanted predictability & cookie cutter processes

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - 1950s & 1960s at NASA, driven by approaches from the 1940s
  - “Program development by stepwise refinement” - Wirth 1971
- The focus of the increments was a little different than today
- And then software development as an industry boomed
  - Companies focused on long term commitments
  - Managers wanted predictability & cookie cutter processes
  - Outsider perspectives drove industry approaches

We Have Learned & Forgotten Much

- Early on, we knew that development was about increments & refinement
  - 1950s & 1960s at NASA, driven by approaches from the 1940s
  - “Program development by stepwise refinement” - Wirth 1971
- The focus of the increments was a little different than today
- And then software development as an industry boomed
  - Companies focused on long term commitments
  - Managers wanted predictability & cookie cutter processes
  - Outsider perspectives drove industry approaches
- Royce’s 1970s paper against monolithic (waterfall) methods was used in support of waterfall....

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s
  - Evolved from the Spiral model in the 1980s

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s
  - Evolved from the Spiral model in the 1980s
  - Varying degrees of planning as necessary for a project

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s
  - Evolved from the Spiral model in the 1980s
  - Varying degrees of planning as necessary for a project
- Contractors were *required* to be agile in order to win contracts

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s
  - Evolved from the Spiral model in the 1980s
  - Varying degrees of planning as necessary for a project
- Contractors were *required* to be agile in order to win contracts
- Industry still enjoys predictability & long term commitments

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s
  - Evolved from the Spiral model in the 1980s
  - Varying degrees of planning as necessary for a project
- Contractors were *required* to be agile in order to win contracts
- Industry still enjoys predictability & long term commitments
  - Break projects into sprints, but plan months out with uncertain requirements

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s
  - Evolved from the Spiral model in the 1980s
  - Varying degrees of planning as necessary for a project
- Contractors were *required* to be agile in order to win contracts
- Industry still enjoys predictability & long term commitments
  - Break projects into sprints, but plan months out with uncertain requirements
  - Forecasting is viewed as committed schedule

We Have Learned & Forgotten Much

- Agile methods in the 1990s reintroduced incremental & refinement based approaches, catching on in the 2000s
  - Evolved from the Spiral model in the 1980s
  - Varying degrees of planning as necessary for a project
- Contractors were *required* to be agile in order to win contracts
- Industry still enjoys predictability & long term commitments
  - Break projects into sprints, but plan months out with uncertain requirements
  - Forecasting is viewed as committed schedule

- This is not a secret
- Good developers & clients are not fooled

What really characterizes agile?

- Key characteristics

What really characterizes agile?

- Key characteristics
  - ***continuous feedback***
    tests, customer guidance, bug reports, prototypes

What really characterizes agile?

- Key characteristics
  - ***continuous feedback***
    tests, customer guidance, bug reports, prototypes
  - ***adaptation***
    adjust short & long term approaches given a problem

What really characterizes agile?

- Key characteristics
  - ***continuous feedback***
    tests, customer guidance, bug reports, prototypes
  - ***adaptation***
    adjust short & long term approaches given a problem
- Process can be good(!), but when it impedes these, it is a problem.

What really characterizes agile?

- Key characteristics
  - ***continuous feedback***
    tests, customer guidance, bug reports, prototypes
  - ***adaptation***
    adjust short & long term approaches given a problem
- Process can be good(!), but when it impedes these, it is a problem.
  - This is ***why*** prototyping trumps planning

What really characterizes agile?

- Key characteristics
  - *continuous feedback*
    tests, customer guidance, bug reports, prototypes
  - *adaptation*
    adjust short & long term approaches given a problem
- Process can be good(!), but when it impedes these, it is a problem.
  - This is ***why*** prototyping trumps planning
- It is problematic enough that the DoD released guidelines on detective “Agile BS”

Some DoD points on agility

- Key ***good*** indicators for developers
  - Automated: testing (unit & regression), security scanning, deployment
  - Full CI/CD pipeline and infrastructure as code
  - Direct feedback from users & client visible issue tracking
  - Issue triage & assignment policies
  - Clear release cycle planning

Some DoD points on agility

- Key ***good*** indicators for developers
  - Automated: testing (unit & regression), security scanning, deployment
  - Full CI/CD pipeline and infrastructure as code
  - Direct feedback from users & client visible issue tracking
  - Issue triage & assignment policies
  - Clear release cycle planning
- ***Red flags***
  - Users are not ***continuously*** able to try the product & provide feedback
  - Meeting a requirement has priority over getting feedback
  - Absence of DevSecOps

Some DoD points on agility

- Key ***good*** indicators for developers
  - Automated: testing (unit & regression), security scanning, deployment
  - Full CI/CD pipeline and infrastructure as code
  - Direct feedback from users & client visible issue tracking
  - Issue triage & assignment policies
  - Clear release cycle planning
- ***Red flags***
  - Users are not ***continuously*** able to try the product & provide feedback
  - Meeting a requirement has priority over getting feedback
  - Absence of DevSecOps

Beware!
Additional buzzwords!

Agility at the development level

- Feedback & adaptation also guide “code construction”

Agility at the development level

- Feedback & adaptation also guide “code construction”
  - Don’t overcommit to a detailed design (different than feature)

Agility at the development level

- Feedback & adaptation also guide “code construction”
  - Don’t overcommit to a detailed design (different than feature)
  - Use designs that tolerate & support change

Agility at the development level

- Feedback & adaptation also guide “code construction”
  - Don’t overcommit to a detailed design (different than feature)
  - Use designs that tolerate & support change
  - Focus on the known behaviors & refine the details as they become apparent

Agility at the development level

- Feedback & adaptation also guide “code construction”
  - Don’t overcommit to a detailed design (different than feature)
  - Use designs that tolerate & support change
  - Focus on the known behaviors & refine the details as they become apparent
- These are very related to the integration strategies from Code Complete & Pragmatic Programmer! [Ch.7, Tip 68]

Agility at the development level

- Feedback & adaptation also guide "code construction"
  - Don't overcommit to a detailed design (different than feature)
  - Use designs that tolerate & support change
  - Focus on the known behaviors & refine the details as they become apparent
- These are very related to the integration strategies from Code Complete & Pragmatic Programmer! [Ch.7, Tip 68]
  - Top Down
  - Bottom Up
  - Sandwich
  - Risk Based

Agility at the development level

- Feedback & adaptation also guide "code construction"
  - Don't overcommit to a detailed design (different than feature)
  - Use designs that tolerate & support change
  - Focus on the known behaviors & refine the details as they become apparent
- These are very related to the integration strategies from Code Complete & Pragmatic Programmer! [Ch.7, Tip 68]
  - Top Down
  - Bottom Up
  - Sandwich
  - Risk Based

What are the risks & benefits of these?

Dealing with incomplete modules

- All of these approaches may integrate components that *do not yet exist*!

Dealing with incomplete modules

- All of these approaches may integrate components that *do not yet exist*!

What strategies do you use to work around this?

Dealing with incomplete modules

- All of these approaches may integrate components that *do not yet exist*!
- Partial, fake, & prototype implementations are common approaches to ensure progress.
  - Just take care that the fake does not become production

Dealing with incomplete modules

- All of these approaches may integrate components that *do not yet exist*!
- Partial, fake, & prototype implementations are common approaches to ensure progress.
  - Just take care that the fake does not become production
- Stub or fake implementations also aid in partitioning and team development!

Dealing with incomplete modules

- All of these approaches may integrate components that *do not yet exist*!
- Partial, fake, & prototype implementations are common approaches to ensure progress.
  - Just take care that the fake does not become production
- Stub or fake implementations also aid in partitioning and team development!
  - First design core API

Dealing with incomplete modules

- All of these approaches may integrate components that *do not yet exist*!
- Partial, fake, & prototype implementations are common approaches to ensure progress.
  - Just take care that the fake does not become production
- Stub or fake implementations also aid in partitioning and team development!
  - First design core API
  - Independent work happens on different “physical” files

Let’s try it out (quickly)