Who are we?

- Who is CollabNet ScrumCORE (formerly Danube)?
  - On site process and technical consulting
  - ScrumWorks project management tool
    - free Basic version
    - inexpensive Pro version*
  - Agile process and technical training

- Who am I?
  - http://ScrumMasterChecklist.org
  - possibly wrote software for the plane that brought you here

* http://danube.com/scrumworks

Technical Debt Agenda

- Concepts
- Practice
**Running (and Tested) Features**

*(Technical Debt is High Cost of Future Change)*

![Diagram showing Agile Theory vs Waterfall Theory]


---

**Discuss Definitions**

- **“Technical Debt”**
  - Metaphor introduced by Ward Cunningham in 1992
  - Usually describes increasing effort for code changes as design entropy occurs
  - Question: does “technical debt” describe *known* undesired behavior?
    - e.g. specifically known bugs/limitations?

- **“Cost”**
  - of making changes
    - e.g. time/effort/money working around sloppy code
  - of unknown behavior (unknown bugs/limitations)

---

**At Which Step Was Debt Incurred?**

1. Araine 4 control system programmers wrote code to convert acceleration measurements from a 64-bit floating point number to a 16-bit integer.
2. Range check and exception handler intentionally omitted, to help meet requirement for CPU load below 80%.
3. Araine 4 software passed design and code reviews.
4. Araine 4 software passed tests for expected operational conditions (acceleration never exceeded 16-bit range).
5. Araine 4 made 113 successful launches.
6. System designers reused Araine 4 control system “as is” for a new vehicle with much stronger acceleration: Araine 5
Define “Cost” of Technical Debt?

- Value of vehicle + payload: $370 million
- Next Ariane 5 launch attempt: 1 year later.
- Rebuilt Cluster Mission payload finally launches with Russian Soyuz rocket: 4 years later
  - First scientific discoveries 2 years after that
- Ariane 5 gains reputation for reliability: 9 years later

Question: What was the actual cost?

Option 1: Code And Fix Anarchy
**Option 2: “Relay Race” Phased Model**

"Chris Peters, Microsoft’s Vice President in charge of Office, also claimed that Microsoft Project is more appropriate for managing the design of airplanes and buildings, rather than individual tasks like software development." -- Michael Cusumano*

In 1970, Dr Winston Royce published "Managing the Development of Large Software Systems", where the waterfall is first documented. He said “I believe in this concept, but the implementation described above is risky and invites failure.”

* Michael Cusumano, Microsoft Secrets (1998), Free Press

---

**Option 3: Rigorous Iterations**

*Inspect/Adapt at Fixed Boundaries*

---

* We know less about the project today than at any time in the future.
  -- Chet Hendrickson


Scrum Overview

Sprint

Sprint

Sprint

Sprint

Sprint

Product

Product

Product

Product

Stabilization Sprint

Shippable Product!

Robust Definition of “Done”

The Potentially-Shippable Product Increment

Each Sprint should yield a potentially-shippable product increment*. If Product Owner said “Ship what you showed me,” would you be ready within one stabilization Sprint?

* "Potentially shippable" does not always mean potentially sellable." -- Jeff Heinen, QPass Inc.

Weak Definition of “Done”

Stuff we procrastinated:
refactoring
load testing
security testing
user documentation
technical debt
UAT...

Shippable Product?
**Running (and Tested) Features**

![Diagram showing the comparison between Scrum and Waterfall methods in terms of technical debt accumulation over time.](image)


**Technical Debt Mudball**

1. **Legacy System**
2. **Legacy System**
3. **Legacy System**

- Malfunctioning code.
- Low quality code lacking automated test.

**Discuss: Seen This Before?**

- Cutting corners to achieve a higher velocity and meet impossible timelines leads to build up of unmaintainable code.
- Death spiral: As the maximum velocity of system goes down, even more corners are cut to compensate until the velocity approaches 0.
- Key people move on to make new messes elsewhere.

*Ref: [http://danube.com/blog/kanemar/technical_debt_and_the_death_of_design_part_1.html](http://danube.com/blog/kanemar/technical_debt_and_the_death_of_design_part_1.html)*
Can Cost of Change be Flattened?

![Cost Over Time Graph]

Preventing Technical Debt

- Keith Braithwaite has been measuring Cyclomatic Complexity per function (a rough indicator of bugs and cost of change) on large amounts of open source Java code. Measurements suggest the best code had one thing in common about the way it was developed. Guess what that was.
  - hint: this answer isn’t “Scrum” or “Kanban”

Preventing Technical Debt

- Define Test-Driven Development.
- Define Refactoring.
  - When should it occur?
- Define continuous integration.
- How can teams improve these practices?
**Definition of “Done”**

**Write Explicitly on Each Item**

- Example engineering criteria to prevent Technical Debt
  - pair programming, code/design review
  - manual test
  - automated test coverage
    - unit tests
    - system tests
      - prefer same language (e.g. Java, not brittle capture/playback or proprietary scripting languages)
  - refactoring
    - changing internals without changing behavior
    - incrementally remove duplicate code, business logic in your presentation layer (JHTML, JSPs, etc.), complex conditional logic, poor naming, obsolete libraries....
    - impractical without automated test

* [http://danube.com/blog/michaeljames/junit_is_not_just_for_unit_testing_anymore](http://danube.com/blog/michaeljames/junit_is_not_just_for_unit_testing_anymore)

---

**Definition of “Done”**

**Write Explicitly on Each Item**

- Business Criteria often forgotten
  - degree of feature richness
  - usability
  - performance
    - timing,
    - scalability,
    - reliability,
    - etc.
  - cross-cutting concerns
    - compliance with corporate integration needs
    - external regulations (i.e. legal)
  - whether/when regression failures allowable

---

**Continuous Integration**

**(Continuous Automated Testing)**

* Toyoda Type G Automatic Loom
Red Green Refactor

• Start by writing a simple failing test

```java
public void testMiddleAgeMale() {
    Person michael = new Person(42, MALE);
    LifeSpan michaelLifeSpan = new LifeSpan(michael);
    assertEquals(1716, michaelLifeSpan.remainingWeekends());
}
```

TEST FAILS (or won’t compile)

Red Green Refactor

• Write code until test passes

```java
public class LifeSpan {
    public LifeSpan(Person person) {
        this.person = person;
    }
    public int remainingWeekends() {
        int x = 365; // TODO: what about leap years?
        boolean daylightSavingsTime = false;
        String whatIsYourFavoriteColor = "blue"; // No, red!
        return 52 * Math.abs(75 - person.getAge());
    }
}
```

TEST PASSES

Red Green Refactor

• Refactor (improve code/design without altering its behavior)

```java
public class LifeSpan {
    public LifeSpan(Person person) {
        this.person = person;
    }
    private int remainingYears() {
        return Math.abs(75 - person.getAge());
    }
    public int remainingWeekends() {
        return 52 * remainingYears();
    }
}
```

TEST PASSES
Red Green Refactor

- Write another simple failing test

```java
public void testNewbornFemale() {
    Person sophia = new Person(0, FEMALE);
    LifeSpan sophiaLifeSpan = new LifeSpan(sophia);
    assertEquals(4160, sophiaLifeSpan.remainingWeekends());
}
```

TEST FAILS

testNewbornFemale expected 4160, got 3900

Red Green Refactor

- Add/change code until test passes

```java
private int remainingYears() {
    if (person.isMale())
        return Math.abs(75 - person.getAge());
    else
        return Math.abs(80 - person.getAge());
}
```

TEST PASSES

Examples of When to Refactor

- Duplicate code
- Complex conditional logic
  - deeply nested if/else statements
    - indented
      - off the right side of the screen
        - like this
          - and this
          - and this...
    - exception handlers inside other exception handlers
- long methods
- poorly named identifiers
- poor cohesion/coupling
- too many responsibilities in one object
**System Test vs. Unit Test**

- Failing systems are often made of working units
- System tests can also be automated, ideally in same language as production system.

```java
compTester.actionDrag(
    srcTable,
    new JTableLocation(backlogItemRow,
                        getBacklogTitleColumnIndex(srcTable)),
    InputEvent.BUTTON1_MASK);

return dragAndDrop(destTableLocation, true);
```

**Sprints Comprise a Release**

**Potentially-Shippable vs. Sellable**

![Diagram showing Sprints Comprise a Release]

**Exercise: Sprints**

**In The Definitions of Scrum**

- Is there such a Sprint as an “analysis Sprint”?
- Is there such a thing as a “testing Sprint”?
- What is a “stabilization Sprint”?
- If someone says a project requires a lot of infrastructure and architecture work that will take eight weeks to complete, should the first Sprint be eight weeks long?
  - Is the architecture an adequate deliverable?

- What does YAGNI stand for?
Modern Technical Practices

Building

- Given an organization working on one product with three development teams and one maintenance team:
  - How many code branches should be maintained?
  - How often should code be checked in?
  - How frequently should builds run?
  - How frequently should unit tests run?
  - How frequently should system tests run?

Example Implementation Roadmap

1. Implement Scrum skeleton (cross-functional teams, monthly deliveries, etc.)
2. Start measuring velocity. (e.g. story points per iteration)
3. Co-locate team members. (ideally in team rooms)
4. Simplify/agilify source code configuration management practices.
5. Begin using continuous integration. (Detect merge/build failures immediately.)
6. Build “sandboxes.” (Independent environments for TDD.)
7. Develop habit of test-driven development. (Detect regression failures immediately.)
8. Develop habit of radical refactoring, continuous design. (Reduce impact of changes to requirements changes.)
9. Assess and scale.

Technical Debt Repayment

- Apply TDD to incrementally repair touch points as new features added.

Silver Bullet?

- Does Scrum prevent Technical Debt?
- Does Big Up Front Design prevent Technical Debt?

Unknown Behavior is Technical Debt

- Not tested since last code change? Assume it is broken.

Technology Problem, or People Problem?

- “Poor management can increase software costs more rapidly than any other factor.”
  - Barry Boehm
**Who are we?**

- **Who am I?**
  - mj4scrum@gmail.com

- **Who is CollabNet ScrumCore?**
  - On site process and technical consulting
  - ScrumWorks project management tool
    - free Basic version
    - inexpensive Pro version*
  - Agile process and technical training

* [http://scrumworks.com](http://scrumworks.com)