REVERSE ENGINEERING PRODUCT LINES IN AGILE ENVIRONMENTS

Jane Cleland-Huang, PhD
Software Product Line Conference, Seville, Spain 2017
JaneClelandHuang@nd.edu
Department of Computer Science and Engineering
University of Notre Dame

Some of the work described in this talk was funded by the US National Science Foundation under Grants CCF-0959924 and CCF-1265178.
What I’m going to talk about..
Focus on constructing and releasing a real-world, evolving safety-critical development environment to support Requirements Engineering research.

This is a system for controlling and coordinating the flights of Unmanned Autonomous Systems in an urban airspace.

Supports both physical UAS and high-fidelity simulators.
Team Work

Michael Vierhauser
Architect

Jane Wyngaard
Hardware

Jinghui Cheng
Post Doc / UI

James, Patrick, Michelle, Joshua Huseman
MS student

Sean Bayley
PhD Student

Alex Madey
Undergraduate

Quinlan McMillan
Undergraduate

Greg Madey
Frank
**Requirements Engineering researchers** interested in using Dronology to support research into traceability, forensic requirements, goal modeling, runtime adaptation....

**End users** interested in deploying Dronology to support specific scenarios of use.

Stakeholders’ needs ultimately drive feature prioritization and variability points.
Dronology: Our Context and Challenge
Dronology: Our Crashes

- Trajectory challenged, upside down drone
- Bent prop drone
- Rescued Drone
- Broken leg drone
- Drowned and missing drone
It could be worse..
It could be worse..
Or potentially damaging
Most of the time..
A platform for coordinating the flight of UAVs. Supports research in safety assurance, runtime monitoring, & adaptation.

*Developed by the Notre Dame Team: Michael Vierhauser, Jane Wyngaard, Jinghui Cheng, Sean Bayley, Greg Madey, Joshua Huseman, Jane Cleland-Huang, & more...*
Where two worlds meet..

A Product Line
A portfolio of related products

Agile Development
A lightweight software process centered around iterative development
Can we merge them?

Software Product Line Engineering

• Upfront planning
• Predict commonalities & variabilities
• Mature processes
• Early architecture design
• Platform adaptability to accommodate future variabilities
• Formal documentation

Agile Development

• Just enough planning
• Requirements (user stories) needed now
• No futuristic designs
• Face-to-face informal communication

Both aim to handle changing requirements
Integrating achieved by applying agile practices within small teams assigned to deliver each product.
Our Goal was different...

We started with an agile process that allowed us to incrementally add features.....

and we wanted to continue with an agile process in order to progressively add variability points..
Agile: XP Style

1. On-site customer
2. User stories
3. Short cycles
4. Acceptance tests
5. Pairs programming
6. Test-driven
7. Collective ownership
8. Continuous integration
9. Sustainable pace
10. Open workspace
11. The planning game
12. Simple design
13. Refactoring
14. Metaphor
EARS: Easy Requirements Specifications

<optional preconditions> <optional trigger> the <system name> shall <system response>

- Simple structure adds rigor & clarity
- System response describes what the system must actually do that is visible at the boundary of the system

1. **Ubiquitous**
   Requirement is always active

2. **Event-driven** (keyword When)
   Required response to a triggering event

3. **State-driven** (keyword While)
   Required response in a specified state

4. **Option** (keyword Where)
   Applicable only if feature is included

5. **Hybrid**: Use combinations of when, while and where for requirements with complex conditional clauses.

EARS “Stories”

**Ubiquitous**

The `<component name>` shall `<response>`

The drone shall maintain a minimum-separation distance at all times.

**Event Driven.**

When `<trigger>` the `<system name>` shall `<system response>`

- **When** the drone is within X centimeters of minimum separation distance from another drone, the collision avoidance system shall provide directives to all drones in the vicinity.

**State Driven**

While `<in a specific state>` the `<system name>` shall `<system response>`

- **While** in landing mode the drone shall descend vertically until it reaches the ground.

**Option**

Where `<feature is included>` the `<system name>` shall `<system response>`

Where parachute mode is enabled and a drop is initiated the drone shall scan the dropzone for obstacles.

**Unwanted Behavior**

If `<optional preconditions>` `<trigger>`, then the `<system name>` shall `<system response>`

- If wind gusts exceed desired wind velocity but are below the maximum wind velocity, the drone shall return to base.
Fault Mode Effect Criticality Analysis (FMECA)

Preliminary Hazard:
Drone’s battery fails unexpectedly and the drone crashes.

1. The battery level detector fails to detect a low battery level.
2. Battery level indicator fails and an incorrect battery level is reported.
3. The software fails to check battery level in time to take responsive actions before the battery fails.
"Safety" Scrum

1. Inputs from Stakeholders, Customers, Users, Team

2. Safety Analysts

3. Preliminary Hazard Analysis

4. Product Owner

5. Preliminary FMECA (Fault Mode Criticality Analysis)

6. Team selects user stories for the sprint

7. FMECA for stories placed into sprint backlog and for new feature interactions

8. Product Backlog

9. Ranked list of what is required – Features, User stories, Safety-Stories

10. Sprint meeting

11. Incrementally Constructed Safety Assurance Case

12. Working Product

Is it currently safe?

Summer REUs UI Development-2017
Reverse Engineering
What does the literature say?

Static analysis

Runtime trace

Topic modeling

But fail to fully leverage human knowledge of the ongoing project.
Architecture: Intuitive, visual, component-based perspective. To what extent do components map to features & variability points?

Requirements: Mapped To epics. To what extent do epics map to features and their variability points?

Source Code: Identify packages. Do they map to features?

All of these have “clues” about possible features...
Features & Variability Points:
What are the variability points?

Elicit stakeholder’s needs. Perform domain analysis.

A fully top-down approach could imply a break-away from agile.
Clues for Commonalities and Variabilities in Jira

- Simulator
  - UI flight route planning view
  - Safety Related Issues
  - Collision Avoidance
  - UI real-time flight view
  - Supporting artifacts
  - Ground Station
  - Ground Station Middleware
  - UAV Hardware
  - Flight route management
  - Core Dronology - UAV management
  - Manage core dronology functionality
  - Manage interface between Dronology and GUI clients
  - Platform, Language, and Portability issues
  - Vaadin

- Large body of work, contains stories
- Small unit of work representing a piece of functionality
- Smallest unit of work.

Epics
- Large body of work, contains stories
- Epics with obvious potential for variability
- Epics representing commonalities

User Stories
- Small unit of work representing a piece of functionality

Subtasks
- Smallest unit of work.

Places that we deliberately built for variability

User Stories
- Small unit of work representing a piece of functionality

Subtasks
- Smallest unit of work.

Epics
- Large body of work, contains stories
- Epics with obvious potential for variability
- Epics representing commonalities
As-Is Product (Jira)

The world as Jira sees it!
As-is Product (Jira + Packages)

As is product!

Augmented by a code-level package view.
Adding a new feature is really an extensive form of impact and untangling analysis. Treat it as a user story with sub-tasks or as an epic.
When variability is not envisioned...

Step 1: Evaluate the feature for its *tangling*.

Step 2: Envision the future.

Step 3: Write untangling stories.
Put the user story and subtask into the backlog.

• Define collision avoidance interface
• Extract concrete collision avoidance implementation to a concrete strategy class.
• Push code shared across collision avoidance algorithms to a common class.

Tasks:
- Define collision avoidance interface
- Extract concrete collision avoidance implementation to a concrete strategy class.
- Push code shared across collision avoidance algorithms to a common class.

1 As a researcher I want to exchange collision avoidance algorithms. (when?)

2 When CollisionAvoidance_HotSwap event occurs, the collision avoidance strategy shall be exchanged at runtime.
What new safety requirements are introduced as a result of the new variability point?

Perform safety hazard analysis on hot swapping collision avoidance algorithms.
Adding a new Feature

UI Middleware
- Coulombs
- Advanced
- Dronology UI Elements
  - Flight Planning
  - Real-time flight view
  - UAV registration

Collision Avoidance
- Virtual Fleet Manager
- Physical Fleet Manager
- Air Traffic Control

Fleet Manager
- Simple
- Nvec

Air Traffic Control
- Flight Plan
- Flight Zone Manager

Simulator

Vehicle
- Physical
- Virtual

Flight Manager

Runtime Monitor

Ground Station Middleware
- ArduPilot Ground Station

Middleware

Dronology UI Elements

UAV registration

Real-time flight view

Flight Planning

Runtime Monitor

ArduPilot Ground Station
Architecturally Significant Variability Points

Architecturally significant requirements are:

- Technically challenging or constraining.
- Have a pervasive impact on the system.
- Often, but not always, related to -ilities

How disruptive is it? Can we do it?
Adding an architecturally significant variability point

- Collision Avoidance
- Fleet Manager
- Air Traffic Control
- Simulator
- Flight Manager
- Flight Plan
- Flight Zone Manager
- Takeoff Manager
- Ground Station Middleware

Is it a real requirement? Where does it come from?
Is it a real requirement?

When do we invest in it?
Next steps

Now we have a product line we need traceability support between many different artifact types
- To perform impact analysis
- To generate safe products
How do we configure and maintain an Agile PL?

**Individuals** and interactions over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

**Almost anything** over heavy-weight traceability

That is, while there is value in the items on the right, we value the items on the left more.
Traceability in Agile SPLs

Based on an analysis of 53 papers which referenced product-line traceability. Only showing links that were mentioned more than one time.

Goal: Leverage agile tracing practices without adding additional overhead.
Traceability in Agile SPLs

Goal: Leverage agile tracing practices without adding additional overhead.
How do we achieve this in an agile way?

Tag commits – and infer trace links from them.

Fill in the gaps by recommending missing tags.

Evolve trace links automatically.

Try to build traceability into the daily process to the fullest extent possible.
TLE: How it works

1. Software artifacts changed across versions

2. Tools for detecting changes in code and requirements.

3. TLE tools and algorithms for recognizing change patterns and evolving trace links.

- Requirements $R_i$
- Source Code $C_i$
- Trace Links $T_i$

- Version $i$

- Requirements $R_{i+1}$
- Source Code $C_{i+1}$
- Trace Links $T_{i+1}$ Evolved

- Version $i+1$

- srcML
- Java-Call Graph
  - Source Code Processing

- Refactoring Crawler
- Similarity Computations using IR Methods

- Change Scenario Detection Heuristics
  - Change Scenarios S1-S18 detected

- Trace Link Heuristics Applied against Scenarios S1-S24 and Trace Links $T_{i+1}$ Generated
Identify Patterns of change

Two classes are merged. The original link is reassigned to the merged class.

Mona Rahimi, Jane Cleland-Huang: Patterns of co-evolution between requirements and source code. RePa@RE 2015: 25-31
Detecting a Scenario

① Use information retrieval techniques, refactoring crawlers, and other static analysis tools to determine which properties hold.

② Infer which change scenario was enacted.

③ Update links

S3 The new class represents a merging of existing ones. Reassign any trace links that associated with the original classes to the merged class.
In our experiments the effort required by humans to confirm or deny TLE links was minimal – with few decision points per day.
A Retrospection on factors that impacted our decisions

Made an explicit decision to favor agility.

Considered uncertainty, volatility, and velocity.

Made timeline decisions based on multiple factors:
-- time to “market”
-- difficulty
-- current workload
Some open research challenges

- Merging mining software repositories (jira and github etc) with static analysis, runtime traces, and topic modeling to identify potential features.
- Providing automated help for transforming tangled code into a variability point (either static or runtime).
- Human Computer Interaction: Creating a dashboard to serve as a burndown chart to show current progress (technical debt) towards achieving variability points.
- “Real” impact analysis where I can create a ‘user’ story defining a new variability point – and the analysis identifies point of impact and difficulty of transforming the code and supporting artifacts.
Dronology Project @ Notre Dame

A platform for coordinating the flight of UAVs. Supports research in safety assurance, runtime monitoring, & adaptation.

Developed by the Notre Dame Team: Michael Vierhauser, Jane Wyngaard, Jinghui Cheng, Sean Bayley, Greg Madey, Joshua Huseman, Jane Cleland-Huang, & more…
Any questions?
REVERSE ENGINEERING PRODUCT LINES IN AGILE ENVIRONMENTS

Jane Cleland-Huang, PhD
Software Product Line Conference, Seville, Spain 2017
JaneClelandHuang@nd.edu
Department of Computer Science and Engineering
University of Notre Dame