CompSci 230
Software Design and Construction

Software Quality 2015S1
Agile approach to testing (XP)

Lecture plan

Week 1:	*No class - Anzac Day* What is software quality? Some key developer practices (version control, testing).
Week 2:	Black box testing. White-box testing. Myers' testing principles.
Week 3:	Traditional approach to testing (Waterfall). Agile approach to testing (XP). Famous failures.

Myers Ch. 9, pp. 179-187
www.agilemanifesto.org

Learning Goals for Today

- Have a working understanding of Extreme Programming (XP).
  - Given a description of a software development process, discuss its conformance with XP principles.
  - What are the major arguments for XP? Against it?
  - Name, and briefly describe, some of the 12 core practices of XP.

- Have a working understanding of testing in XP.
  - Given a description of a software testing process, discuss its conformance with XT.
  - What are the major arguments for XT? Against it?
  - Given a development scenario, discuss whether XT could be applied. If it is applicable, what benefits might be expected from its use?

Extreme Programming (XP)

- Last lesson, we introduced the agile methodologies as a reaction to traditional software development approaches.

- Today, we will focus on one of these, Extreme Programming (XP).
- In particular, we will discuss testing in an XP context.

A Brief History of XP

- “XP was invented in 1996,
  - when Kent Beck, a software developer,
  - was called in by an American car maker, Chrysler,
  - to rescue a project which had proved so frustrating that it had been scrapped.
- “As Mr Beck worked on this benighted venture,
  - known as Chrysler Comprehensive Compensation (C3),
  - he formulated a set of directions for keeping code ‘elegantly written’.
- “The C3 system now provides
  - correct monthly payroll information for more than 86,000 employees.
  - Its success is ascribed to Mr Beck's golden rules.” [*The Economist*, 9 December 2000]

XP Values (in1998)

- "Extreme Programming rests on the values of *simplicity, communication*, *testing*, and *aggressiveness*."
  - "Extreme programmers do the simplest thing that could possibly work."
  - "Extreme programmers leave the system in the simplest condition possible."
    - "Having built these simple objects, we refactor our code to eliminate any redundancy and ugliness in the code just installed."
  - "Customers are part of the team throughout development."
  - "We graph functional test scores every day, showing green for correct results and red for incorrect."
  - "There's no waiting for features needed in some other class, so we move quickly." [The C3 Team, "Chrysler Goes to 'Extremes', *Distributed Computing*, Oct 1998, pp. 24-8]

The Purpose of XP (according to Myers)

- "The purpose of the XP development methodology is
  - to create quality programs in short time frames."
- "Classical software processes still work,
  - but often take too much time,
  - which equates to lost income
  - in the competitive arena of software development."

Basics of XP (according to Myers)

- XP focuses on implementing simple designs,
  - Communicating between developers and customers,
  - Constantly testing your code base,
  - Refactoring to accommodate specification changes, and
  - Seeking customer feedback."
- *? Is this an accurate description of XP?* In their 1998 article, the C3 team described the purpose of refactoring differently:
  - "Having built these simple objects, we refactor our code to eliminate any redundancy and ugliness in the code just installed."
- XP is a general approach to software development, not a specific process
  - Kent Beck (and other "gurus" of XP) have refined their ideas about XP, over the past 16 years.
  - *? When was it decided that XP is a 'general approach' – I'm sure this was not the case at the start?*

http://laerer.rhs.dk/vibekes/4%20sem-sym/articels/TheNewXP.pdf

Strengths of XP (Myers)

- "XP tends to work well for
  - small to medium-size development efforts
  - in environments that have frequent specification changes and
  - where near-instant communication is possible."
- "XP… avoids the large-scale project syndrome, in which the customer and the programming team meet to design every detail of the application before coding begins."
  - "Project managers know this approach has its drawbacks because customer specifications and requirements constantly change to reflect new business rules or marketplace conditions."
  - Myers doesn't point out that customers rarely know "what they want" until they see a prototype of what is possible – then they can discuss what they like, what they don't like, and what new features they want.

Strengths of XP (cont.)

- Myers: “the XP methodology … avoids coding unneeded functionality.”
  - “If your customer thinks that the feature is needed but not required, it generally is left out of the release.”
  - “Thus, you can focus on the task at hand, adding value to a software product.”
  - (This is probably what the C3 team meant by “aggressiveness” in their 1998 article.)

XP in a Nutshell (Myers)

- Four basic concepts:
  - Listening to the customer and other programmers
  - Collaborating with the customer to develop the application's specification and test cases.
  - Coding with a programming partner.
  - Testing the code base.

The Importance of Planning in XP (Myers)

- "The planning phase in XP differs from that in traditional development models, which often combine requirements gathering and application design."
- "Planning in XP focuses on identifying your customer's application requirements and designing user stories (or case stories) that meet them."
  - "You gain a significant insight into the application's purpose and requirements when creating user stories."
  - "In addition, the customer employs the user stories when performing acceptance tests at the end of a release cycle."
  - "Finally, an intangible benefit of the planning phase is that the customer gains ownership and confidence in the application by heavily participating in it."

Examples of User Stories

- As a user, I want to search for my customers by their first and last names.
- As a non-administrative user, I want to modify my own schedules but not the schedules of other users.
- As a mobile application tester, I want to test my test cases and report results to my management.
- Starting Application: The application begins by bringing up the last document the user was working with.
- As a user closing the application, I want to be prompted to save if I have made any change in my data since the last save.

[Source: http://en.wikipedia.org/wiki/User_story]

Use Case Diagrams

- XP uses informal, brief stories.
- An alternative style of informal specification is the Use Case Diagram (as discussed earlier this term).
  - Use-case diagrams were invented by Ivar Jacobsen in 1986 for use in his "Objectory Process".

12 Original Practices of XP

1. Planning and requirements ("user stories"; customer chooses features)
2. Small, incremental releases
3. System metaphors
4. Simple designs
5. Continuous testing
6. Refactoring
7. Pair programming
8. Collective ownership of the code
9. Continuous integration (every day)
10. 40-hour work week
11. On-site customer ("you and your programming team have unlimited access to the customer so you may resolve questions quickly and decisively")
12. Coding standards ("all code should look the same")

Testing is Central in XP

- "The XP model relies heavily on unit and acceptance testing of modules.
  - The philosophy is "extreme".
  - Maximise the main defect-finding activities, stop other testing activities.
  - ? *Do you agree that these are the main testing activities? What about system testing? load testing? Do these become part of acceptance testing? Or is acceptance testing only apply to functionality? If the former, doesn't this put a huge load on the customer?*

Testing is Central in XP

- “The XP model relies heavily on unit and acceptance testing of modules.
- “In general, you must run unit tests for every incremental code change,
  - no matter how small,
  - to ensure that the code base still meets its specification.”

Testing is Central in XP

- "In fact, testing is of such importance in XP that the process requires that you
  - create the unit (module) and acceptance tests first,
  - then create your code base.
  - this form of testing is called, appropriately, *Extreme Testing* (XT)."
  - *? Isn't a more common name "Test Driven Development" (TDD) ?*
  - *? Did Kent Beck really invent the idea ?*
  - *? Was TDD originally part of XP ?*

http://www.drdobbs.com/extreme-testing/184414994
http://arialdomartini.wordpress.com/2012/07/20/you-wont-believe-how-old-tdd-is/

Extreme Unit Testing

- According to Myers, XUT has two "simple rules":

1. All code modules must have unit tests before coding begins
2. All code modules must pass unit tests before being released into production.

- Nothing new here, except the insistence on writing unit tests before coding.
  - Wow, that's a disciplined approach! Would you do this willingly, or would you be tempted to "code early, on the sly" (when you think your manager isn't looking)?
  - Note: if you're writing executable tests, then I'd say you're coding.
    - You can write assertions in Java or Junit.
    - You might be programming in a language with goal-directed evaluation (e.g. Icon): specify the outcome (in restricted settings) and let the computer figure it out!
    - In a futuristic/AI development scenario, you could be "programming by example" i.e. goal-directed programming without any sharply-defined restrictions.

Benefits of "Test-First Coding" (Myers)

1. You gain confidence that your code will meet its specification.
2. You express the end result of your code before you start coding.
   - *? Why does Myers think this is a benefit?*
3. You better understand the application's specification and requirements.
4. You may initially implement simple designs and confidently refactor the code later to improve performance [*and elegance – important for maintainability*] without worrying about breaking the specification.
   - *? OK, so I shouldn't "worry" about the specification when I refactor, but don't I have to consider it?*

The "Shining Point" of XP

- "The practice of creating unit tests first is the shining point of the XP methodology, as it forces you to understand the specification to resolve ambiguities *before* you begin coding."
  - ?Really? What keeps me from writing unit tests hastily, without resolving ambiguities?
- Myers' reasoning isn't clear to me but YMMV (your mileage may vary ;-)
  - "… you may not fully understand the acceptable data types and boundaries for the input values of an application if you start coding first.
  - "So how can you write a unit test to perform boundary analysis without understanding the acceptable inputs?
  - "Can the application accept only numbers, only characters, or both?
  - "If you create the unit tests first, you *must* understand the specification."
- My translation: "If you create a robust set of black-box tests, this demonstrates that you understand the specification. Only after demonstrating this, should you start to code." (I'm in the "quality school" of testing)
  - So… there must be some quality-control on the unit tests.
  - What's the main activity in this quality control? (We'll see Kent Beck's answer later.)

The importance of automated testing

- "Manually running unit tests, even for the smallest application, can be a daunting task.
- "As the application grows, you may generate hundreds or thousands of unit tests.
- "Therefore you typically used an automated testing suite to ease the burden of constantly running unit tests."
- Main functions of an automated testing suite:
  - Script the tests, then run all or part of them
  - Create reports and classify the bugs: may be useful in future development
- The "testing code base" becomes as valuable as the software application itself, so it should be
  - stored in a code repository,
  - with adequate backups & security

Extreme acceptance testing (XAT)

- Purpose of XAT: to determine, with a minimum of effort and time, whether the application is acceptable to the customer.
  - Is XP unsuitable for use whenever there is more than one customer?
  - Some software development methodologies (as taught in information-systems departments) acknowledge that
    - stakeholders are often deeply conflicted about requirements on new IT systems, and
    - IT system specifications are "levers of change" in an organisation, implying that
    - Acceptance-testing decisions by an employee may be "over-ruled" by management.
- Let's assume the XAT team has a fully-empowered and well-informed "customer" at their disposal! Then…
  - "… customers, not you or your programming partners, conduct the acceptance tests."
  - "In this manner, customers provide the unbiased verification that the application meets their needs."

Relation of XAT to user stories

- "Customers create the acceptance tests from user stories."
  - "The ratio of user stories to acceptance tests is usually one to many.
  - "That is, more than one acceptance test may be needed for each user story."
  - *? This sounds like quite a burden on the customer! I wonder how many XP projects are actually doing this?*

Automation of XAT?

- "Acceptance tests in XT may or may not be automated.

- "For example, an unautomated test is required when the customer must validate that a user-input screen meets its specification with respect to color and screen layout."
  - Hmmm… a layout & colour test could be fully automated, with some image-processing techniques, if the specification is very precise. However a spec that's a one-line "user story" is will require some subjective measurement of attributes such as "easy to use", "legible", "attractive".

- "An example of an automated test is when the application must calculate some payroll values using data input via some data source such as a flat file to simulate production values."

XAT: a Validation or a Verification?

- In XAT, as described by Myers, the customer is asked whether or not the system produces valid output.
  - System validation: commonly defined as "Are we building the right thing for you?"
  - System verification: "Are we building it right?", that is, does the system meet its specifications?
  - Requirements validation: "Do the requirements specify a system that you, the customer, would want to use – assuming we can build it?"
  - Requirements verification: "Do the requirements make sense?", that is, could our dev team understand them well enough to implement them, or are they too vague, contradictory, or infeasible?

- Because customers aren't allowed to change their stories during an acceptance test, I'd say XAT is a system verification.
  - Any new stories should be prioritised into the release schedule.
  - *My question: If there are no new stories during XAT, is the current system valid?*

The Peril of Changing Requirements

- If previously-accepted requirements are changed, the development may make little or no "forward progress"
  - Developers must revise unit tests, and then recode the units, to conform to the new requirements

- Trish Koo (http://trishkhoo.com/2009/01/extreme-testing-xt/):
  - "Whenever requirements change, the developers adapt fairly easily but the testers are still gritting their teeth because they have to update a huge backlog of test cases for regression testing."

- If previously-accepted requirements are "cast in concrete", then the project may fail as soon as the "mistake" is discovered.
  - In my (very limited) experience, end-users rarely know what they want until they have fiddled with a prototype. Then the stories change rapidly!

Software updates are hazardous

- When software is in the field, it is **very** hazardous to change any of its features.
  - This is true even when a feature is reported as a "bug" by some stakeholders, and the QA team agrees that it is a bug.
  - Other stakeholders may have become accustomed to the buggy behaviour, and are likely to be confused, annoyed, or even angered when it is "fixed".
- In a sports analogy, this hazard is called
  - "changing the rules after the game has started".
- My advice: unless there's a major security risk, a major legal risk, or a major dissatisfaction among stakeholders with a software product,
  - User-visible behaviour ("look and feel") should remain constant.
  - Even during a major version-step, feature-change should be minimised.
  - I say this because I believe most users don't want to learn new features, adjust their behaviour, or modify their expectations… but … if a change is "really cool" then users will happily "invest" significant time and money in order to gain a novel experience!

Managing stakeholder conflicts in XAT

- XAT (and XP) has no explicit process for managing stakeholder conflict.
  - XP assumes the devteam will have good access to an "on-site customer"
    - To learn more, see http://agilemodeling.com/essays/activeStakeholderParticipation.htm.
  - Most XP projects have a primary stakeholder: the organisation that commissions the software.
    - But see Grünbacher & Hofer, "Complementing XP with Requirements Negotiation", in *Proc. 3rd Int'l Conf. on eXtreme Programming and Agile Processes in Software Eng'g*, 2002, available http://cf.agilealliance.org/articles/system/article/file/909/file.pdf.

- Who could be a valid "on-site customer" for an XAT on `createAppletImageIcon()`?
  - If they're a professor, they'll have a hard time understanding the student point of view.
  - If they're a student, they'll have a hard time understanding the prof's point of view.,
  - If they are a prospective future user, they might have an entirely different point of view.

- The IBM Rational Unified Process, and many other "IS-style" development processes, treat stakeholder conflict as a first-order concern.
  - I think there can be no "magic bullet": stakeholder conflict is a fundamental problem in software development.

Myers’s Summative Evaluation of XP

- “Although glamorous, XP is not for every project or every organization.”
- “Proponents of XP [claim] that the chances of successful application development increase dramatically”
- “Detractors say that because XP is a process, you must do all or nothing. If you skip a practice, then … your program quality may suffer.”
- “[D]etractors claim that the cost of changing a program in the future to add more features is more than the cost of initially anticipating and coding the requirement.”
- “[S]ome programmers find working in pairs very cumbersome and invasive, therefore they do not embrace the XP methodology.”

Is C3 a Success Story for XP?

- *"The original estimate done by the C3 team in March 1996 was that the project would be ready to ship in about a year.*
  - *It launched in about a year.*
  - *I think it was about two months later than was wanted owing to a late understanding of what the Customer needed for testing.*
- *"The launch was considered a success by everyone.*
- *"Subsequent launches of additional pay populations were wanted by top management within a year.*
  - *The team thought that was possible… After two? more years the next group was ready to ship in the team's opinion but something always got in the way.*
  - *It wasn't quite like the 90% done syndrome, but there was always another requirement that just had to be done.*
  - *Communication up and down the chain of command was broken; every manager but one on both the IT side and Finance side was replaced or moved to a new position.*
- *"Finally the project was terminated.*
  - *At this writing [2002], C3 is no longer paying any employees, though it did so until the end of 2000.*
- *"Was it a process failure? It's hard to say.*
- *"The things that XP deals with were all chugging along, but*
  - *it was as if the project had become uninteresting to the high-level stakeholders, and*
  - *they forgot about it and then one day remembered and turned it off."*

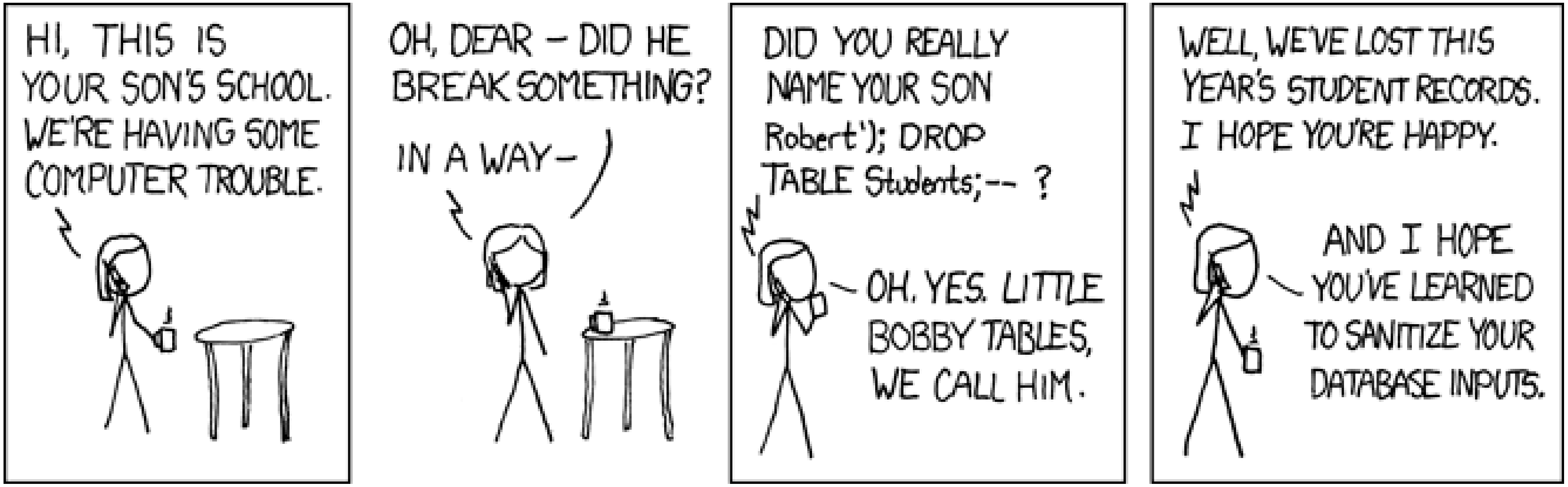
Source: http://c2.com/cgi/wiki?ChryslerComprehensiveCompensation, retrieved 15 October 2013.

Is C3 a Success Story for XP?

- C3 was started in January 1995 with a 26-man team.
  - After a year and a half, the project had hit a brick wall …
  - Reportedly, the development team had lost sight of its goal of printing checks.
  - Also, no good way of testing was in place.
- In March 1996, Kent Beck took over the project.
  - In talking one-on-one with each team member, he had [basically] laid out the ground rules for extreme programming (XP), which he then applied to C3.
- Then he announced that in the short time of three weeks they would be printing out their first check.
  - The team was surprised at the announcement, since they had just spent eighteen months and not printed anything.
  - They made that goal.
  - Next, it was onto the other 86,999 checks that needed to be printed.
  - By August 1998, C3 was paying about 10,000 people.
- The project was cancelled in January/February 2000 [because]
  - it was only paying one-third of Chrysler employees,
  - the Y2K period had passed and
  - the mainframe software was still operating correctly, and
  - the project was over budget.

Source: Harvey Herela, Case Study: The Chrysler Comprehensive Compensation System. Galen Lab, U.C. Irvine, 21 April 2005. Available at http://calla.ics.uci.edu/histories/ccc/ from 2007-2012. See http://web.archive.org/web/20070415000000*/http://calla.ics.uci.edu/histories/ccc/.

Sanitising Inputs: A Humorous View

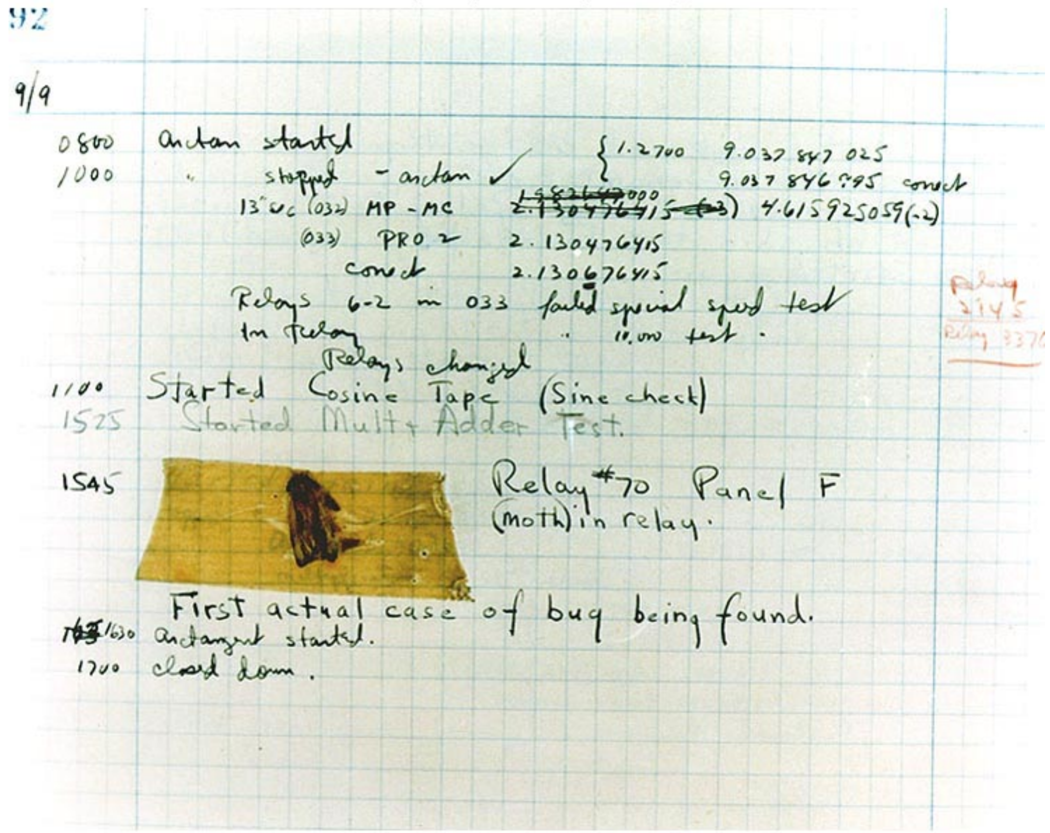

HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY –
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://imgs.xkcd.com/comics/exploits_of_a_mom.png

The First “Computer Bug”

- Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator
  - while it was being tested at Harvard University, 9 September 1945.
- The operators affixed the moth to the computer log, with the entry:
  - “First actual case of bug being found”.
- They put out the word that they had “debugged” the machine, thus introducing the term “debugging a computer program”.
  - In 1988, the log, with the moth still taped by the entry, was in the Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia.

Photo # NH 96566-KN (Color) First Computer “Bug”, 1947



92

9/9

0800 Anctan started
1000 " stopped - anctan ✓ { 1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In Relay " 11.000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult+ Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Anctangent started.
1700 closed down.

Relay 2145
Relay 3370

Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988.
http://www.history.navy.mil/photos/images/h96000/h96566kc.htm

Where are we now?

- Many development methodologies have been proposed
  - Each tends to gather a number of zealots who:
    - advocate application of the methodology under all circumstances
    - spend significant time 'proving' the methodology works under all circumstances.
  - Architects originally stated that all practices must be carried out:
    - each supports the others
    - if you omit some, there will be gaps in the process

- The current wisdom is that no one approach is guaranteed to be effective in all cases
  - Each organisation must tailor a development process suitable for its specific contexts
  - 'Agile' has morphed to mean 'flexible'

- When the next 'silver bullet' arrives (as it surely will), you should ask 'what is the evidence?'