# Rapid Application Development

CS-370: Software Design and Development

Robert Bruce

# What is Rapid Application Development?

- Rapid Application Development (RAD):
  - An evolutionary software development model.
- Common features in RAD:
  - The number of developers is limited to six or fewer people and scope is limited.
  - Requirements gathering is conducted through “focus groups, workshops, facilitated meetings, prototyping, and brainstorming”.
  - Requirements are validated through “iterated prototypes, use cases, and constant customer testing of designs.”
  - “Repeated customer testing of design as they evolve.”
  - New software is constantly integrated and tested.
  - “Informal reviews and communication among team-members.”
  - Turn-around time for iterations lasts between one-week and a few-months.
  - Complicated features deferred for later release.
  - Tight delivery schedule.

Source: pages 467-468, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# RAD: Advantages

- Promotes accurate requirements based-on customer's needs.
- Provides flexibility with changing requirements.
- Promotes engaged stakeholders.
- Reduces development time.
- Promotes software recycling (code-reuse).
- Promotes early software releases (with limited functionality)
- Promotes constant software testing minimizing software defects and code-migration issues.
- Promotes risk mitigation during each development iteration.
- Promotes likelihood for project success by identifying problematic solutions quickly through frequent project iteration.

Source: pages 468-469, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# RAD: Disadvantages

- The evolutionary RAD model promotes resistance to change by some software developers.
- Inefficient for large projects which require extensive communication and large teams.
- Requires skilled software developers, not beginners.
- “Requires access to scare resources.”
- Requires additional overhead “if the requirements are known completely and correctly in advance.”
- Requires laissez-faire management approach (e.g. reduced managerial control).
- Project results may be sub-optimal.
- Unpredictable: some customers are looking for a completed product by a fixed date rather than interactive development over an unknown period of time.

Source: page 469, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# James Martin’s original RAD model: Four phases

- “Requirements planning”
- “User design”
- “Construction”
- “Cutover”

Source: page 470, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Agile approach: a set of principles

- Agile approach focuses on principles rather than a development model.
- Agile is used for enhancing development models.
- Agile approach emphasizes:
  - Communication
  - Incremental development
  - Focus on quality

Source: pages 471-478, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Agile approach: Twelve principles

1) “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”
2) “Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.”
3) “Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.”
4) “Business people and developers must work together daily throughout the project.”
5) “Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.”
6) “The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”
7) “Working software is the primary measure of progress.”
8) “Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.”
9) “Continuous attention to technical excellence and good design enhances agility.”
10) “Simplicity – the art of maximizing the amount of work not done – is essential.”
11) “The best architectures, requirements, and designs emerge from self-organizing teams.”
12) “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”

Source: page 472, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Extreme Programming (XP) Values

- **Communication** from:
  - All stakeholders on a shared vision for project goals.
- **Simplicity** in design by:
  - Only adding features when necessary.
- **Feedback** from:
  - Customers to developers on usability and development direction.
  - Developers to developers on code quality;
  - Developers to customers on difficulty and time-requirements for product changes.
- **Courage** to focus on:
  - A simple solution despite knowing how to develop more complicated solutions.
  - Refactoring code when necessary.
  - Discarding unnecessary code when necessary.
  - Providing feedback.
  - Giving and receiving respect.

Source: page 480, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Extreme Programming (XP) Practices

- ‘Have a customer on-site”
- “Play the planning game”
- “Use stand-up meetings”
- “Make frequent small releases”
- “Use intuitive metaphors”
- “Keep designs simple”
- “Defer optimization”
- “Refactor when necessary”
- “Give everyone ownership of the code”
- “Use coding standards”
- “Promote generalization”
- “Use pair programming”
- “Test constantly”
- “Integrate continuously”
- “Work sustainably”
- “Use test-driven and test-first development”

Source: page 481, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Extreme Programming (XP) Practices

- ‘Have a customer on-site”:
  - promotes open-communication and shared-values.
- “Play the planning game”:
  - Planning the next release of the product to the customer
  - Planning the next iteration of the product
- “Use stand-up meetings”
  - Conduct team meetings while standing to minimize meeting duration.
- “Make frequent small releases”
  - Give the customer a product as-soon-as-possible
  - Receive customer feedback from the latest release.

Source: pages 481-483, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Extreme Programming (XP) Practices

- “Use intuitive metaphors”
  - Makes communicating product features easy to understand by the customer.
- “Keep designs simple”
  - The interface should only support the immediate task at hand.
- “Defer optimization”
  - Prioritize meeting project requirements over speed-increases.
- “Refactor when necessary”
  - Remove unnecessary code which does not support current requirements.
- “Give everyone ownership of the code”
  - Minimize wasted time waiting for other team members to update code by granting every team member the ability to make changes using a source tree repository with check-in, check-out, and merge capability.

Source: pages 484-485, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Extreme Programming (XP) Practices

- “Use coding standards”
  - Consistent coding standards and conventions makes the software source tree easy to read, update, and manage by the developers.
- “Promote generalization”
  - Encourage developers to learn all areas of the software source tree to better understand how the whole project works as a system and improve development skills.
- “Use pair programming”
  - Write code in real-time with one or two other developers watching. The benefit is immediate discussions about the implementation and quality of the software.
- “Test constantly”
  - Test software frequently and often. Utilize automation in the testing process as well.

Source: page 481, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Extreme Programming (XP) Practices

- “Integrate continuously”
  - The code which comprises a software product should be rebuilt then tested often to ensure interoperability and stability.
- “Work sustainably”
  - Place limits on daily time spent on software development for work-life balance to prevent developer stress and burn-out.
- “Use test-driven and test-first development”
  - Develop in a modular style programming with stubbed-in functions (empty function declarations with no code inside).
  - Develop one-function at a time.
  - Test the newly completed function as well as previous completed functions with inputs before beginning to develop a new code stub.
  - Refactor any completed functions if necessary.

Source: pages 487-488, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Feature-Driven Development (FDD) model

- An iterative and incremental development model.
- FDD was developed for large team projects.
- FDD projects utilize five phases:

  1) “Develop a model”
  2) “Build a feature list”
  3) “Plan by feature”
  4) “Design by feature”
  5) “Build by feature”

Source: pages 500-504, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Feature-Driven Development (FDD) model

- In an FDD project, the first three phases occur at the beginning of the project:
  - “Develop a model”
  - “Build a feature list”
  - “Plan by feature”
- The last two phases are iteratively developed to project completion:
  - “Design by feature”
  - “Build by feature”

Source: pages 500-504, *Beginning Software Engineering* (2nd edition) by Rod Stephens