Documentation, reporting in agile projects

Agile documentation

- From agile manifesto:
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
- Documentation is not related to project success
- Related agile roles: all, technical writers

Agile documentation

Agile documentation

- Minimizing documentation is primarily good
  - Extreme case: all documentation is waste
- Documentation
  - is a requirement like any other, just another task
  - is collaborative, written by the whole team
  - captures high level information, details are waste of time
  - should be just good enough
  - should be ready just-in-time

Documentation – depth

- Documentation should be simple: KISS, DRY principles
- Documentation does not need to be perfect
- Fewest documents, least overlap, information at the most appropriate place

Documentation – types

- Contract model – technical interface for other teams
- Design decisions – optional, for the future
- Requirements – optional, for large or distributed teams
- Executive overview, project overview, reports – for the management
- System – high level architecture, requirements
- Support – training material, troubleshooting
- User – API, user guide, support guide, training

Documentation – efficiency

- Document stable concepts, the “document late” practice
  - Not what we “plan to do”, but what we “did”
  - Documentation needs not to be updated
  - No waste of time on speculative ideas
  - The documentation is several iterations behind the development – documentation gap
- Most information is available in the tests that specify
  - requirements
  - architecture
  - design
- Documentation generation from source code

Documentation – what

- Each document
  - must have a purpose: process with no customers → no documentation
  - should focus on the needs its readers: comprehensive documentation is rarely needed
  - must have meaning and provide value: no value → no documentation
- What
  - critical information
  - good things to know
  - do not include obvious information or what the user is supposed to know

Documentation – when

- When to document
  - ideally at each iteration
  - when it helps the communication of participants, with external groups and with stakeholders
  - when a model has to be kept up-to-date
  - when it hurts, but just-in-time
- When to update
  - With each new release
  - When it hurts, leads to loss of productivity
- Yes, documents may not be consistent

Documentation – issues

- Software vs. documentation development: documentation does not provide new functionality
- Developers are not technical writers: when to hand over the task
- Documentation needs to be refactored as well, high level documents are easier to update
- Who is the audience? Self-documenting code is not enough

Documentation – output format

- Publishing tools: XML based, Wiki based
- Source documentation
- Reports, audit: charts for supporting management decisions

Self-documenting code

- Assembly is for computers, source code is for programmers
- Self-documented code does not replace documentation
- Follow the conventions of the agile project
- Principles:
  - Commenting is waste of time – in most cases
  - Don’t write code that is hard to understand
  - Comment only code pieces that are hard to understand
  - Comments become out-of-date during refactoring

Self-documenting code

- Basic methods:
  - Naming: names of functions, variables, constants should explain purpose
  - Extracting functions: identify purpose of a snippet
  - Introduce variable: describe expression with a variable name
  - Code grouping: move related lines in same snippet
  - Interfaces: identify the set of exposed functions
  - Exception instead of TODO comment

Self-documenting code

- Naming:
  - Active word and subject: sendFile
  - Indicate return value
  - Avoid words like: “manage”, “make”, “handle”
  - Indicate units: widthPx
  - Avoid parameter names: a, i, s
  - Named constants instead of values: maintained at a single place

```
public static final int THE_ANSWER = 42;
```

Self-documenting code

- Extract functions
  - Move a snippet (e.g. a step of an algorithm) to a separate function to clarify purpose
  - The helper function may be reused

```
width = (value – 0.5) * 16;

width = emToPixels(value);
…
int emToPixels(float ems) {
    return (ems – 0.5) * 16;
}
```

Self-documenting code

- Introduce variable:
  - Complex relational expressions within conditions may not convey the intent of the expression

```
is_cold = temperature < 0;
if (is_cold) { … }
```

  - Clarify complex expressions

```
(x – x0)*(x – x0) + (y – y0)*(y – y0)
horizontalError = (x – x0)*(x – x0)
verticalError = (y – y0)*(y – y0)
horizontalError + verticalError
```

Self-documenting code

- Throw NotImplementedException instead of TODO comments
- Grouping:
  - Put related lines in the same snippet to signal which lines must be maintained together

```
foo = 1;

blah();
xyz();

bar (foo);
baz(1337);
quux(foo);
```

```
foo = 1;
bar (foo);
quux(foo);

blah();
xyz();

baz(1337);
```

Self-documenting code

- Interfaces
  - More meaningful function names

```
class Box {
  public void setState(int state) {
    this.state = state;
  }
  public int getState() {
    return this.state;
  }
}
```

```
class Box {
  public void open() {
    this.state = 1;
  }
  public void close() {
    this.state = 0;
  }
  public boolean isOpen() {
    return this.state == 1;
  }
}
```

Refactoring techniques

- Extracting
  - Component/Module/Service/Class:
    - one component is doing the job of two
    - same behavior in more components
  - Interface: more components with the same set of behavior
  - Subcomponent: features used only in some instances
  - Supercomponent: same attributes in several components
  - (Private) Method: grouped code fragment

Refactoring techniques

- Inlining
  - Component/Module/Service/Class:
    - no duplication prevented
    - a component is not doing much
  - Method: naming does not make the behavior clearer
  - Temporary variable: used once and naming of the right hand side of the definition is clear

Refactoring techniques

- Moving
  - Field: used in another component more frequently
  - Method: uses fields of another component more frequently
- Pull up in component hierarchy
  - Constructor: same body
  - Field: same field in subcomponents
  - Methods: identical behavior in subcomponents
- Push down in component hierarchy
  - Field: specific to subcomponents
  - Method: relevant to subcomponents

Refactoring techniques

- Replacing
  - Array with object
  - Hash with object
  - Field with association
  - Constructor with factory
  - Inheritance with delegation, delegation with inheritance
  - Temporary variable with call chains
- Change association directions

Documentation generation

- Code documentation is still necessary for the reference guide (API)
- Document what a function does, but not how it does that
  - lists of usage
  - side effects
  - possible return values
  - algorithms must not be documented

Documentation generation

- Lots of tools available: e.g. doxygen, JavaDoc
- Specially commented source
  - Usually /** */ comments and @ prefixed tags inside
- Output: usually web page

Reporting

- For informing product and project managers about the
  - progress of development
  - quality metrics
  - technical debt
- Report generation is a scheduled task
  - Email notification

Reporting

- What does a report consist of?
  - Data model object: software quality metrics collected automatically by static code analysis tools, test results
  - Layout objects: how to visualize the data (charts, tables)
  - Parameters: how to configure the visualization
  - Scripts: that gather the data from the source code, logs and test results
  - The code itself to show the location of a technical debt

Reporting

- Available tools for CI
  - SonarQube
  - Lots of others