LOG8430: Architecture logicielle et conception avancée

Fundamentals of Software Architecture

Automne 2017

Fabio Petrillo
Chargé de Cours

Page 2

Page 3

What is a Death March project?

Page 4

Page 5

5

Page 6

6

Page 7

HealthCare.gov Application Crash (“ObamaCare”)

- USA Federal Health Insurance
- Company
  - CGI Group Inc. (Montreal/Canada)
  - CGI was a key contractor
- Failure
  - problematic launch - application poor performance
  - 2013
- Impact
  - The program does not achieve its objective of facilitating health insurance services for **millions of Americans**
  - American **election** results???

Venkatesh, V., Hoehle, H., Aljafari, R. (2014). "A usability evaluation of the Obamacare website". Government Information Quarterly. Volume 31, Issue 4, 2014, Pages 669-680

Page 8

Death March Defined

A Death March is a project is destined to fail

In most software death march projects, this usually means one or more of the following constraints has been imposed:

- The **schedule** has been compressed to **less than half the amount of time**

estimated by a rational estimating process

- The budget and associated resources have been cut in half
- The staff has been reduced to less than half the number of people that would normally be assigned to a project of this size and scope
- The **functionality**, features, performance requirements, or other technical aspects of the project **are twice** what they would be under normal circumstances and/or **discussed previously (change of requirements)**

8

Page 9

"ObamaCare" Application Crash - Causes

- **integration/infrastructure issues (Architectural issues in fact)**
- Lots of subcontractors (47!!!)
  - who was ultimately responsible???
- ~60k user sim -> 250k (naïve)
- Large requirements
- "Big Bang" approach
- Huge pression
- Lack of **integration** testing

"We **didn't** see full **end-to-end testing** until a **couple of days** leading up to the launch"

http://adage.com/article/digitalnext/lessons-user-experience-healthcare-gov/244933

http://www.nytimes.com/2013/10/25/us/politics/bipartisan-dismay-over-health-plan-woes-at-house-hearing.html

Page 10

Why does it happens?

Page 11

11

Page 12

Politics,

politics, politics.

Page 13

Death March - Why it happens (more)? (1997)

- **Naive promises** made by marketing, senior executives, naive project managers, etc.
- **Naive optimism** of youth: "We can do it over the weekend!"
- The **"start-up"** mentality of fledgling, entrepreneurial companies.
- The "Marine Corps" mentality: **Real programmers don't need sleep**
- Intense competition caused by **globalization** of markets.
- Intense competition caused by the appearance of **new technologies**.
- Intense pressure caused by **unexpected government regulations**.
- **Unexpected** and/or unplanned **crises** — e.g., your hardware/software vendor just went bankrupt, or your three best programmers just died of Bubonic

Plague.

13

Page 14

So, how to address that problems? What could we do?

14

Page 15

There is **no way to rescue**
Death March Projects!
Quit as quick as possible,
if it is possible.

Page 16

Fundamentals of Software Architecture

Page 17

17

Page 18

18

Page 19

Software architecture (ANSI/IEEE Std 1471-2000)

*Architecture is defined by the recommended **practice** as the fundamental **organization** of a system, embodied in its **components**, their **relationships** to each other and the **environment**, and the principles governing its **design** and **evolution**.*

Page 20

Software architecture (SEI definition)

*The **software architecture** of a computing system is the set of **structures** needed to*

reason about the system, which comprise software elements, relations among them, and properties of both.

Page 21

Software architecture (Sangwan, 2015)

Architecture is fundamentally concerned with the **organization** of a system into its constituent **elements** and their **interrelationships** to **achieve a given**

purpose.

21

Page 22

Software architecture

- Software architecture emerged as an subfield of software engineering explicit in the 1990s
- array of proposals for notations, tools, techniques, and process naturally as a by-product of good design to support architectural design
- **software architecture has in many cases not found its way into common practice.**
- Two main schools about the role that architecture
  - **architecture-focused design**: focus on detailed and **complete** architectural designs
  - **deemphasizes architecture**: it will **emerge** naturally as a by-product of good design; naturally as a by-product of good design
- Clearly, **neither** of these camps has it right for **all systems**.

22

Page 23

Partitioning, knowledge, and abstractions

- Three intellectual "**weapons**"
- **Partitioning**: a strategy to combat **complexity** and scale when two conditions are true: first, the divided parts must be **sufficiently small** that a person can now solve them; second, it must be possible to **reason** about how the parts assemble into a whole.
- **Knowledge**: use knowledge of **prior problems** to help developers solve current ones.
- **Abstraction**: Abstraction effectively can combat complexity and scale because it **shrinks problems**, and smaller problems are easier to reason about.

Page 24

System design = architecture + detailed design

*"As the size and **complexity** of software **increase**, the design and specification of overall system **structure** become more significant issues than the choice of algorithms and data structures of computation. Structural issues include the organization of a system as a composition of components; global control*

structures; the protocols for communication, synchronization, and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; and ***selection among design alternatives****. This is software architecture level of design."*

Shaw and Garlan, 1996

Page 25

Software architecture

- Software Architecture is the **macroscopic design** of a software system
- Software architecture is about the **design impact** of your system and the it has on the system's qualities, qualities like performance, security, and modifiability.
- Architecture acts as the **skeleton** of your system, influences its **quality attributes**, and constrains the system.
- It is mostly **orthogonal** to the system's functionality

Page 26

Three examples of software architecture

- Rackspace is a real company that managed hosted email servers. Use case: customer queries
- Rackspace built three generations of systems to handle the customer queries
- Version 1: Local log files -> **grep** on the servers
- Version 2: Central database -> **single** machine to receive queries and operations. Several problems!
- Version 3: Indexing cluster
  - **ten machines and Hadoop**
  - saving log data into a distributed file system and by parallelizing the indexing of log data.
  - Rackspace was able to index over 140 gigabytes of log data per day and had executed over 150,000 jobs since starting the system.
- Same functionality (queries) -> three different architectures

Page 27

Discussion in pairs (or more)

What did they gain and lose with each architectural version?

Page 28

Just Enough Architecture

- **How much architectural design should one carry out for a given system?**
- Fairbanks (2010) argues that the **core criterion** for determining how much

- architecture is enough is **risk reduction**
- How one defines "fundamental risks" depends on the **stakeholders'** perspectives on the system goals and should at least be complete with respect to the stakeholders' concerns

Page 29

Stakeholders

29

Page 30

Applying software architecture ideas

- Edsger Dijkstra - “GOTO Considered Harmful” (1968)
- GOTO statements vs structured programming
- Mindset changing

- *“Effectively applying software architecture ideas requires a conscious and explicit shift to embrace its abstractions, such as components and connectors, rather than only using the abstractions found in mainstream programming language (often just classes or objects).”*

30

Page 31

Characteristics of software architectures

- Architecture defines **structure**: how to sensibly partition an application into a set of interrelated components, modules, objects or whatever unit of software partitioning works for you, minimizing dependencies between components, and creating a loosely coupled architecture from a set of highly cohesive components.
- Architecture specifies component **communication**
- Architecture addresses **nonfunctional** requirements
  - **Technical** constraints: “We only have Java developers, so we must develop in Java”.
  - **Business** constraints: “In order to widen our potential customer base, we must interface with XYZ tools”
  - **Quality** attributes: an application’s requirements in terms of scalability, availability, ease of change, portability, usability, performance, and soon.

Page 32

Architecture Elements

- The **static structures** of a system define its internal design-time elements and their arrangement.
- The **dynamic structures** of a system define its runtime elements and their

interactions.

Page 33

Architecture is most important when...

- the failure **risks** are high (ObamaCare).
- there is a **small solution space** or it is hard to design any acceptable solution.
- stakeholders demand difficult **quality attributes**
- when you are working in a **new domain** (for you)
- Architecture can help **manage complexity**
- The value of creating a **flexible** architecture is the cost of not addressing the technical aspects that can help a system **adapt to changes** when they happen.
- Advice: many systems can **succeed** even when their developers **ignore**

software architecture, but plenty of **failures** that could have been **avoided** by **paying attention** to software architecture

33

Page 34

Software architecture

34

Page 35

Characteristics of software architecture

- **Architecture acts as the skeleton of a system**: Every system has an architecture, whether its developers consciously chose it or not.
- **Architecture influences attributes quality**: Quality attributes are externally visible properties, such as security, usability, latency, or modifiability.
- **Architecture is (mostly) orthogonal to functionality**. It is possible to build the same system as a 3-tier architecture or as a peer-to-peer system. There is no single best architecture, but architectures are more suited to some tasks than others.
- **Architecture constrains systems**. Constraints like these act as **rails** and are essential in the construction of a system, in its ability to perform its job, and in the ability to **maintain it over time**. Constraints promote integrity conceptual and reduce complexity, transferring wisdom or understanding from one developer to another.

35

Page 36

Presumptive architectures

- A **presumptive** architecture is a family of architectures that is dominant in a particular domain in a period of time
- No one will be fired for using that.

- Low mental cost
- Examples
  - 60-80: Centralized (mainframe) systems
  - 90: Client/Server
  - 2000: N-tier (MVC)
  - 2006: Service Oriented Architecture
  - 2007: Ajax/Single Page Application
  - 2010: Microservices

Page 37

Three approaches to software architecture

- **Architecture-indifferent or *ad hoc* design**: pay little attention; architecture may emerge without a conscious choice or guided by a presumptive architecture or following a corporate standard.
  - **Systems that follow presumptive architectures usually succeed.**
  - Suited to low-risk projects (Ex. Indie games).
- **Architecture-focused design**: design an architecture that is suitable to achieve your goals quality, which include functionality and attributes. You must be on the lookout for requirements that will influence your architecture choices.

- **Architecture hoisting**: developers design the architecture with the intent of guaranteeing a goal or property of the system. Once a goal or property has been hoisted into the architecture, developers will not need to write any additional code to achieve it.

Page 38

Discussion

What approach do you apply usually?

Page 39

How much architecture work should you do?

- Architecture and design efforts should address the **failure risks** you perceive
- Solve problems tricky using models, because models simplify the problems.
- It is difficult to decide how much of the system's functionality you should model.

Page 40

What does a software architect do?

Page 41

http://dilbert.com/strip/2017-07-18

41

Page 42

Software architect - a classical vision

- Liaison: Architects play many liaison roles. They liaise between the customers or clients of the application and the technical team, often in conjunction with the business and requirements analysts.
- Software Engineering: Excellent design skills are what get a software engineer to the position of architect. They are an essential prerequisite for the role. More broadly though, architects must promote good software engineering practices. Their designs must be adequately documented and communicated and their plans must be explicit and justified. They must understand the downstream impact of their decisions, working appropriately with the application testing, documentation and release teams.
- Technology Knowledge: Architects have a deep understanding of the technology domains that are relevant to the types of applications they work on.

42

Page 43

- Risk Management: Good architects tend to be cautious. They are constantly

Architects architecting architectures

- **The job role: architect**. One possible job title (or role) in an organization is that of a software architect. Some architects sit in corner offices and make pronouncements that are disconnected from the engineering reality, while other architects are intimately involved in the ongoing construction of the software. Either way, the title and the office are not intrinsic to the work of designing or developers, building software. All software not just architects, should understand their software's architecture.
- **The process**: architecting. The process that the team follows is separable from the design that emerges. nearly it is impossible to tell what architecting process followed by a team looking only at the finished software.
- **The engineering artifact**: the architecture. Every software system has an architecture just as every car has a design.

Page 44

Conceptual framework for system and software architecture

Page 45

Architecture Views

- 4+1 View Model (Krutchen, 1995)
- SEI “Views and Beyond” approach
- Rozanski, Woods [2005]

Page 46

4+1 architectural view model [Kruchten 1995]

- Describing the architecture of software-intensive systems, based on the use of multiple, concurrent views
- It is not possible to capture the functional features and quality properties of a complex system in a single comprehensible model that is understandable by and of value to all stakeholders.
- **View**: A view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders.

Kruchten, Philippe (1995, November). Architectural Blueprints — The "4+1" View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.

Page 47

4+1 architectural view model [Kruchten 1995]

Kruchten, Philippe (1995, November). Architectural Blueprints — The "4+1" View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.

Page 48

4+1 Views: Scenarios

- The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They also serve as a

starting point for tests of an architecture prototype.

- This view is also known as use case view.
- Use case diagram, activity diagram, or business process modeling (BPM)

Page 49

4+1 views: Logical view

- The logical view is concerned with the functionality that the system provides to end-users.
- This describes the architecturally significant elements of the architecture and the relationships between them. The logical view essentially captures the structure of the application using class diagrams or equivalents.
- UML Diagrams used to represent the logical view include Class diagram, Activity diagram, State Diagram.

Page 50

4+1 views: Process view

- The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc.
- This focuses on describing the concurrency and communications elements of an architecture. In IT applications, the main concerns are describing multithreaded or replicated components, and the synchronous or asynchronous communication mechanisms used.
- UML Diagrams to represent process view include the Activity diagram.

Page 51

4+1 views: Development view

- The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view.
- This captures the internal organization of the software components, typically as they are held in a development environment or configuration management tool. For example, the depiction of a nested package and class hierarchy for a Java application would represent the development view of an architecture.
- UML Diagrams used to represent the development view include the Component diagram and Package diagram.

Page 52

4+1 views: Physical view

- The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This

view is also known as the deployment view.

- This depicts how the major processes and components are mapped on to the applications hardware. It might show, for example, how the database and web servers for an application are distributed across a number of server machines.
- UML Diagrams used to represent physical view include the Deployment diagram.

Page 53

53

Page 54

Stakeholder

54

Page 55

Page 56

Risks due to multiplicity of views

- Wrong views
- Fragmentation

- Inconsistency
- Kruchten, Rozanski-Woods provide guidance but specifics of the system should be taken into account
- Each view has to be created and maintained costs!
- Views have to be kept consistent, the more views the more potential inconsistencies costs!

Page 57

(Serebrenik , 2012)

Page 58

Architecture Rationale

(Serebrenik , 2012)

Page 59

SEI “Views and Beyond” approach

- **Module**: This is a structural view of the architecture, comprising the code modules such as classes, packages, and subsystems in the design. It also captures module decomposition, inheritance, associations, and aggregations.
- **Component and connector**: This view describes the behavioral aspects of the architecture. Components are typically objects, threads, or processes, and the connectors describe how the components interact. Common connectors are sockets, middleware like CORBA or shared memory.
- **Allocation**: This view shows how the processes in the architecture are mapped to hardware, and how they communicate using networks and/or databases. It also captures a view of the source code in the configuration management systems, and who in the development group has responsibility for each modules.

Page 60

Is UML important?

Nugroho, 2014 - The impact of UML modeling on defect density and defect resolution time in a proprietary system

<< The results confirm that not only does the production of UML class diagrams and sequence diagrams possibly help improve the quality of software, but also it possibly help increase the productivity in software maintenance >>.

Page 61

http://agilemodeling.com/

Page 62

Agile modeling

- Modeling or documenting?

Page 63

Agile modeling - modeling

- Modeling or documenting?
- Modeling = to make models to understand a problem
- Comprehension
- High level of abstraction
- Sketch
- Take a picture and throw in the trash

Page 64

Agile modeling - documenting

- Modeling or documenting?
- Documenting = to make artefacts for communication

- Describe and store/registry decisions
- For “perinity” or “posterity”
- More details
- More structured documents (templates)
- “Formal” modeling notation -> UML, SysML, Archimate, ...

Page 65

Agile modeling session

Page 66

Agile Modeling

- Make essential modeling
- Whiteboard
- Process -> Requirements -> Architecture
- UML notation as base reference -> simple diagrams
- **Make several models simultaneously**
- Diagrams work together (views)
- Decide -> modeling or documenting?

Page 67

Architecture for agile developers

- In their desire to **cut out unnecessary** steps in software development, some agile developers believe they should avoid software architecture techniques.
- This reluctance is **not universal**, as many important voices in agile community support some planned design work, including Martin Fowler, Robert Martin, Scott Ambler, and Granville Miller (Fowler, 2004; Martin, 2009; Ambler, 2002; Miller, 2006). **Refactoring a poor architecture** choice can be prohibitively **expensive** on large systems.
- Just **enough** architecture. The **risk-driven model** of architecture guides developers to do just enough architecture then resume coding.

Page 68

Page 69

Activity - 4+1 View model - Twitter

Using the 4+1 architectural view model, propose and describe the “NeoTwitter” architecture.

- Diagrams
- Agile modeling

Page 70

Discussion sur les articles

- Outils
- Les groups de TPs
- Essayer de choisir un thème
- Plusieurs groups peut choisir le même thème